SRI VENKATESWARA INTERNSHIP PROGRAM
FOR RESEARCH IN ACADEMICS
(SRI-VIPRA)

**Project Report of 2023:**

**SVP-2341**

# "Artificial Intelligence and Machine Learning based analysis of Power Signal Disturbances"

**IQAC**
**Sri Venkateswara College**
**University of Delhi**
**Benito Juarez Road, Dhaula Kuan, New Delhi**
**New Delhi -110021**

**SRIVIPRA PROJECT 2023**

# Title : Artificial Intelligence and Machine Learning based analysis of Power Signal Disturbances

| | |
|---|---|
| **Name of Mentor: Dr. Rahul**<br>**Name of Department: Electronics**<br>**Designation: Assistant Professor** | |

***List of students under the SRIVIPRA Project***

| **S.No** | **Photo** | **Name of the student** | **Roll number** | **Course** | **Signature** |
|---|---|---|---|---|---|
| 1 | | Ayush Sambher | 1721072 | BSc. (H) Mathematics | Ayush |
| 2 | | Sanjh Solanki | 1721064 | BSc. (H) Mathematics | Sanjh solanki |
| 3 | | Ipshita Tripathi | 1621017 | BSc. (H) Electronics | Ipshita |
| 4 | | Abhimanyu Pratap Singh | 1621001 | BSc(H) Electronics | [illegible] |
| 5 | | Kartavya Gupta | 1620016 | BSc. (H) Electronics | Kartavya |

Rahul

**Signature of Mentor**

## Certificate of Originality

This is to certify that the aforementioned students from Sri Venkateswara College have participated in the summer project SVP-2023 titled "**Artificial Intelligence and Machine Learning based analysis of power signal disturbances"**. The participants have carried out the research project work under my guidance and supervision from 15 June 2023 to 15th September 2023. The work carried out is original and carried out in an online/offline/hybrid mode.

[illegible]

**Signature of Mentor**

## Acknowledgements

First and foremost, we thank the Almighty God for giving us the strength, patience, and knowledge that enabled us to successfully complete the project.

We convey our deepest gratitude to the respected Principal of Sri Venkateswara College, Prof. C. Sheela Reddy for conducting the Internship Program and bringing us such an opportunity, and for her unwavering support and encouragement throughout the research process. Their commitment to fostering an environment of academic excellence has been instrumental in the success of this project.

Also, we would like to extend our sincere gratitude to IQAC Coordinator, Prof Swarn Singh and SRIVIPRA Coordinator, Dr Sharda Pasricha for their continuous support throughout the project. Their dedication to promoting research initiative has been indispensable. Their investment in this project allowed the authors to pursue their research goals effectively.

We are grateful to the Department of Electronics for encouraging us to widen our academic perspectives. Their mentorship, insightful feedback, and dedication to academic excellence have greatly enriched the quality of this research.

We owe a deep sense of gratitude to our mentor Dr. Rahul, Assistant Professor, Department of Electronics for his constant support and guidance, it was his encouragement and assistance that made this work possible. In the course of these weeks, we got an insight into the significant algorithms and the concepts of Artificial Intelligence and Machine learning, while learning about several important libraries of Python and working on them. It was an enriching learning experience for us all.

Lastly, we would like to thank all those people who have been associated with this project and have helped us with it. Immeasurable appreciation and deepest gratitude to all of you for your help and support.

## TABLE OF CONTENTS

# Preface

In an era characterized by the relentless pursuit of technological advancement and endless demand for electricity, the quality of electricity supplied to our homes, businesses and industries has become a matter of utmost importance. important. The power we receive must not only be reliable, but must also meet strict quality standards. As we enter a world that increasingly relies on electrically powered devices and systems, understanding and managing complex power quality issues is more essential than ever.

This research report, titled "AI/ML based analysis of power signal disturbances" delves deeper into the intersection of two transformational areas: Power quality analysis and artificial intelligence/machine learning. In these pages, we begin our journey to explore how AI and ML can be harnessed to improve upon the way we assess, monitor and improve electric energy quality. The driving force behind this research is driven by the increasing complexity of power grids, the rise of renewable energy sources, and the growing need for energy efficiency. AI/ML offers a promising solution to directly meet these challenges.

In this report, we dive into the basics of the concepts of AI/ML used for the analysis, the various metrics and parameters that determine power quality, as well as the effects of poor power quality on our power systems. We then set out to explore the field of AI/ML, examining how these technologies can be applied to analyze and predict power quality anomalies with unprecedented accuracy and speed. Throughout our journey, we highlight the real-world applications and benefits of AI/ML-based power quality analytics.

As we publish the findings and insights from our extensive research, we hope this report will be a valuable resource for researchers, engineers, policymakers and stakeholders. related investment in the future of power quality. We believe that through the combination of power quality analytics and AI/ML, we can usher in a new era of energy management that is not only efficient and reliable, but also sustainable in environmental level.

Finally, we express our gratitude to all individuals and organizations who contributed to this study. We hope that the knowledge contained in these pages will inspire further innovation, collaboration and progress in the field of power quality analysis using AI/ML.

# Introduction

In the modern world, electricity has become an integral part of our daily lives, powering everything from our homes and businesses to our transportation systems and industries. . As our dependence on electricity increases, ensuring electricity quality also becomes more important. Power quality refers to the stable and reliable supply of electrical power, free from interference, harmonics, and other unusual phenomena that can negatively impact the performance of electrical devices and systems.

The importance of energy quality is emphasized by its far-reaching implications. Poor power quality can lead to equipment failure, increased energy consumption, reduced lifespan of electrical appliances and even safety risks. Additionally, in an era characterized by the increasing integration of renewable energy sources, electric vehicles and smart grids, maintaining high energy quality becomes even more important to ensure stability. stability and sustainability of our energy system.

Traditionally, power quality analysis has relied on manual methods, which are often time- consuming, resource-intensive, and limited in their ability to handle the complexity of modern power systems. However, with the advent of artificial intelligence (AI) and machine learning (ML) technologies, there is an unprecedented opportunity to improve the way we assess and manage energy quality.

This research report explores the intersection between AI/ML and power quality analytics. We are looking at developing and applying AI/ML techniques for power quality analysis, ultimately leading to the creation of the Power Quality Index (PQI). This PQI is considered a comprehensive measure that captures different aspects of energy quality, providing a more accurate and effective means of assessment than conventional methods.

Our research aims to achieve several main goals:

- Improved accuracy:

By leveraging the capabilities of AI/ML algorithms, we seek to improve the accuracy of power quality assessment, allowing for more accurate identification and characterisation of electrical disturbances and anomalies.

- Real-time monitoring:

We are exploring the potential of AI/ML to enable real-time power quality monitoring, which is critical for timely intervention and maintenance.

- Data-Driven Insights:

Through analysing large data sets, we aim to uncover hidden patterns and correlations in power quality data, providing valuable insights to operators, utilities and power grid researcher.

- Automation and predictive maintenance:

By leveraging predictive analytics, we plan to develop models that can predict power quality problems before they occur, thereby facilitating proactive maintenance and minimise downtime.

- Energy efficiency:

Improving power quality not only ensures the reliability of the power system but also contributes to improved energy efficiency and sustainability, in line with broader environmental goals.

As we begin our journey at the intersection of AI/ML and power quality analytics, we hope that the insights and innovations presented in this report will pave the way for energy infrastructure more flexible, efficient and sustainable. Integrating AI/ML technology promises to not only improve our understanding of energy quality, but also enable us to address the challenges and opportunities in a rapidly changing energy landscape.

## Artificial Intelligence and Machine Learning Concepts:

From healthcare and banking to self-driving cars and language processing, artificial intelligence (AI) is gaining popularity in solving challenging challenges across a range of industries. Extraction of pertinent features from data, optimisation of algorithms to increase effectiveness and accuracy, and selection of acceptable classification approaches are some of the major problems in establishing AI projects. An overview of these crucial components in the context of an AI project is given in this introduction.

The following processes provide an overview about any type of data when processed under the umbrella of Artificial Intelligence:

**Feature extraction**:

The cornerstone of any AI project is feature extraction. It entails locating and deciding on the most instructive components of unprocessed data that are pertinent to resolving a certain issue. In essence, it converts complicated data into a more controllable and insightful representation. An AI system's performance and efficiency can be significantly impacted by effective feature extraction.

**Techniques for optimisation**:

Techniques for optimisation are crucial for optimising the performance of Artificial Intelligence algorithms. These methods guarantee that models converge more quickly, use less resources, and yield more precise results.

**Techniques for classifying data**:

At the core of supervised machine learning are techniques for classifying data into predetermined classes or labels. The type of data and the issue at hand determine which categorisation algorithm should be used.

To meet the project's goals, it will be crucial to combine efficient feature extraction, optimisation strategies, and classification algorithms. These factors work together to support the creation of reliable and effective AI systems that are capable of resolving challenging real-world issues. By utilising these methods, we hope to expand the potential of AI and open up fresh avenues for creativity and judgment.

## Libraries, Modules and Functions:

Many libraries and modules are necessary for data processing, feature extraction, modelling, and assessment when using Python to analyse the Power Quality Index (PQI) for Artificial Intelligence (AI) and Machine Learning (ML) applications. A quick comment on several significant libraries and modules utilised in this context is provided below:

1. **NumPy**: The cornerstone Python module for scientific computation is NumPy. In order to effectively handle and manipulate power quality data, it supports multidimensional arrays and matrices.

2. **Pandas:** Pandas is a strong library for data analysis and manipulation. For organising and cleaning PQI datasets as well as doing exploratory data analysis (EDA), it provides data structures like DataFrames.

3. **Matplotlib and Seaborn:** These visualisation tools are essential for producing illuminating plots and charts that show the distributions, trends, and anomalies of data related to power quality.

4. **Scikit-Learn:** Scikit-Learn is a thorough machine learning package that includes tools for model evaluation, regression, clustering, and classification. It can be used to apply many ML techniques for PQI analysis, including neural networks, decision trees, and support vector machines.

5. **TensorFlow and PyTorch:** These deep learning frameworks are crucial for constructing and

training neural networks for more advanced, high-dimensional power quality data processing. They provide deep learning challenges with adaptability and scalability.

6. **Scipy:** Scipy expands on NumPy and offers more features for technical and scientific computing. It has statistical features used for performing statistical analysis.

Utilising these Python packages and modules will enable you to construct robust AI/ML solutions for PQI analysis and eventually improve power quality management and system stability. These solutions will ease the collection, preprocessing, modelling, and evaluation of power quality data.

## Feature Extraction Algorithms in Machine Learning:

An essential stage in algorithms for machine learning and data analysis is feature extraction. By converting unprocessed data into a more understandable and illuminating representation, it plays a crucial part in improving the effectiveness, accuracy, and accessibility of these algorithms.

Feature extraction essentially serves as a link between unprocessed data and efficient machine learning methods. It makes it possible to change data into a more useful format, which in turn helps the general effectiveness and usefulness of these algorithms in addressing a variety of real-world issues.

Some of the algorithms that we used for feature extraction are:

a. Convolutional Neural Networks (CNNs)
b. Self-Organizing Maps (SOMs)
c. Multi-Layer Perceptron (MLP) – a special type of Feed-Forward Neural Network (FNN)

### Convolutional Neural Networks (CNNs):

CNNs are largely used in computer vision and image analysis. In jobs where convolutional layers are capable of automatically acquiring pertinent features, they may also be employed for feature extraction. The convolutional layers of a CNN may extract abstract and hierarchical features from the initial data in the context of feature extraction. The activations of the final convolutional layer may then be utilized to extract these characteristics, which can subsequently be fed into another machine-learning model. To extract features from pictures before passing them into a machine learning model (a classifier) for a separate task, such as object identification or image retrieval, one may employ pre-trained CNN models

in image classification tasks.
To conclude it all;

a. CNNs are primarily used for image and video recognition tasks.

b. It utilizes specialized layers called convolutional layers.

c. These layers apply filters to input data, enabling the network to learn local patterns and spatial hierarchies.

d. CNNs are often used for tasks like image classification, object detection, and image segmentation.

e. For a clearer visual understanding, check out the accompanying image -

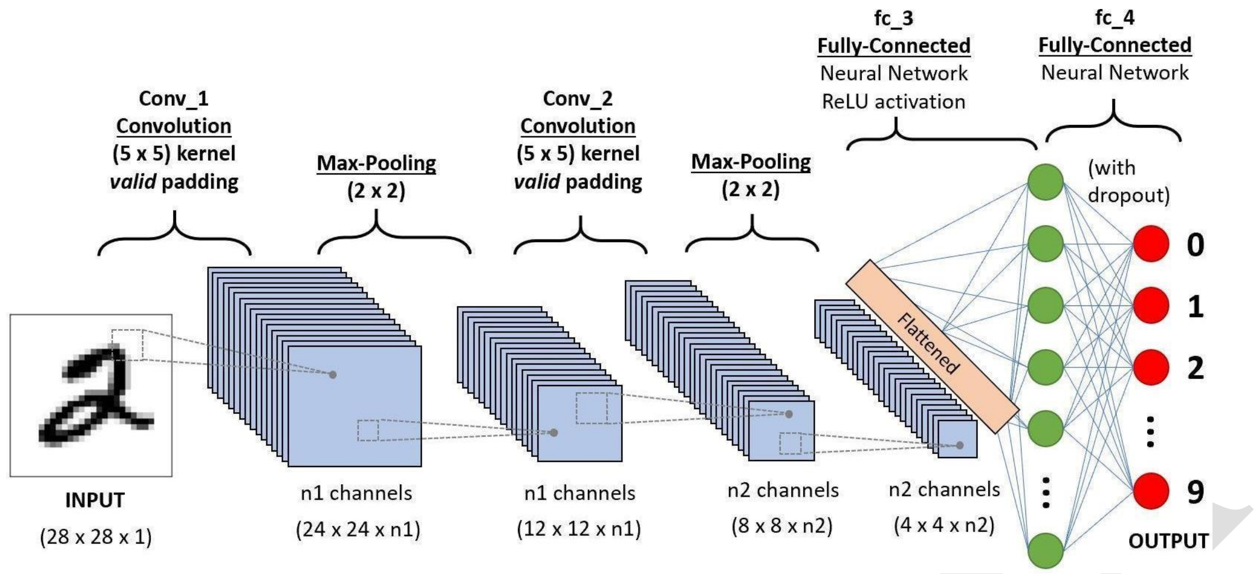


**Fig.1 CNN Architecture**

## Program 01 using CIFAR-10 Dataset:

```
In [3]: import tensorflow as tf
        from tensorflow.keras import datasets, layers, models
        import matplotlib.pyplot as plt
        import numpy as np
```

```
In [4]: (x_train, y_train),(x_test,y_test) = datasets.cifar10.load_data()

        Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
        170498071/170498071 [==============================] - 5140s 30us/step
```

```
In [5]: x_train.shape
Out[5]: (50000, 32, 32, 3)
```

```
In [6]: x_test.shape
Out[6]: (10000, 32, 32, 3)
```

```
In [7]: #So, there are 50000 training images and 1000 test images
```

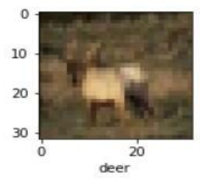
```
In [8]: y_train = y_train.reshape(-1,)
        y_train[:5]
Out[8]: array([6, 9, 9, 4, 1], dtype=uint8)
```

```
In [9]: y_test = y_test.reshape(-1,)
```

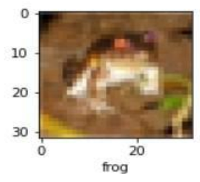
```
In [10]: classes = ["airplane","automobile","bird","cat","deer","dog","frog","horse","ship","truck"]
```

```
In [13]: def plot_sample(x, y, index):
             plt.figure(figsize = (15,2))
             plt.imshow(x[index])
             plt.xlabel(classes[y[index]])
```

```
In [14]: plot_sample(x_train, y_train, 3)
```

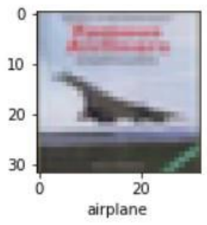


```
In [17]: plot_sample(x_train, y_train, 0)
```



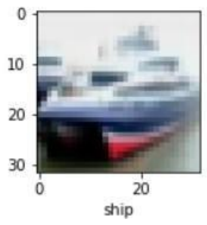
```
In [23]: cnn.fit(x_train, y_train, epochs=10)

         Epoch 1/10
         1563/1563 [==============================] - 44s 27ms/step - loss: 1.4861 - accuracy: 0.4680
         Epoch 2/10
         1563/1563 [==============================] - 42s 27ms/step - loss: 1.1079 - accuracy: 0.6108
         Epoch 3/10
         1563/1563 [==============================] - 42s 27ms/step - loss: 0.9760 - accuracy: 0.6642
         Epoch 4/10
         1563/1563 [==============================] - 42s 27ms/step - loss: 0.8945 - accuracy: 0.6894
         Epoch 5/10
         1563/1563 [==============================] - 44s 28ms/step - loss: 0.8290 - accuracy: 0.7152
         Epoch 6/10
         1563/1563 [==============================] - 42s 27ms/step - loss: 0.7802 - accuracy: 0.7301
         Epoch 7/10
         1563/1563 [==============================] - 43s 28ms/step - loss: 0.7326 - accuracy: 0.7433
         Epoch 8/10
         1563/1563 [==============================] - 45s 29ms/step - loss: 0.6934 - accuracy: 0.7584
         Epoch 9/10
         1563/1563 [==============================] - 43s 27ms/step - loss: 0.6530 - accuracy: 0.7703
         Epoch 10/10
         1563/1563 [==============================] - 43s 28ms/step - loss: 0.6210 - accuracy: 0.7810

Out[23]: <keras.callbacks.History at 0x22cb1ef2eb0>

In [26]: cnn.evaluate(x_test,y_test)

         313/313 [==============================] - 4s 10ms/step - loss: 0.9222 - accuracy: 0.7002

Out[26]: [0.9221606254577637, 0.7002000212669373]

In [31]: y_pred = cnn.predict(x_test)
         y_pred[:5]

         313/313 [==============================] - 3s 10ms/step

Out[31]: array([[1.0070705e-04, 1.6234317e-03, 3.8159089e-04, 8.9445913e-01,
                 3.6065959e-04, 3.2140009e-02, 1.0557044e-02, 3.7585422e-05,
                 5.9130061e-02, 1.2098055e-03],
                [7.5115859e-05, 2.6265301e-03, 2.6336534e-07, 8.6637549e-08,
                 8.2994867e-10, 1.3613261e-08, 3.4995637e-09, 1.1672256e-08,
                 9.9692935e-01, 3.6868150e-04],
                [3.6153108e-02, 9.4817825e-02, 1.3080648e-03, 7.0011890e-03,
                 7.3920359e-04, 5.3691451e-04, 9.8760749e-05, 1.0616910e-03,
                 8.1528348e-01, 4.2999823e-02],
                [6.4228785e-01, 9.8741762e-03, 2.4480576e-02, 2.8516329e-03,
                 7.5042555e-03, 2.9631611e-04, 1.9168808e-03, 2.6204952e-04,
                 3.1026617e-01, 2.6014913e-04],
                [2.2695625e-07, 1.6238599e-06, 1.5016131e-02, 1.3693355e-02,
                 2.9316720e-01, 1.6886687e-03, 6.7591476e-01, 8.7633089e-07,
                 5.1712408e-04, 1.1681114e-07]], dtype=float32)

In [32]: y_classes = [np.argmax(element) for element in y_pred]
         y_classes[:5]

Out[32]: [3, 8, 8, 0, 6]
```

```
In [38]: plot_sample(x_test, y_test,3)
```

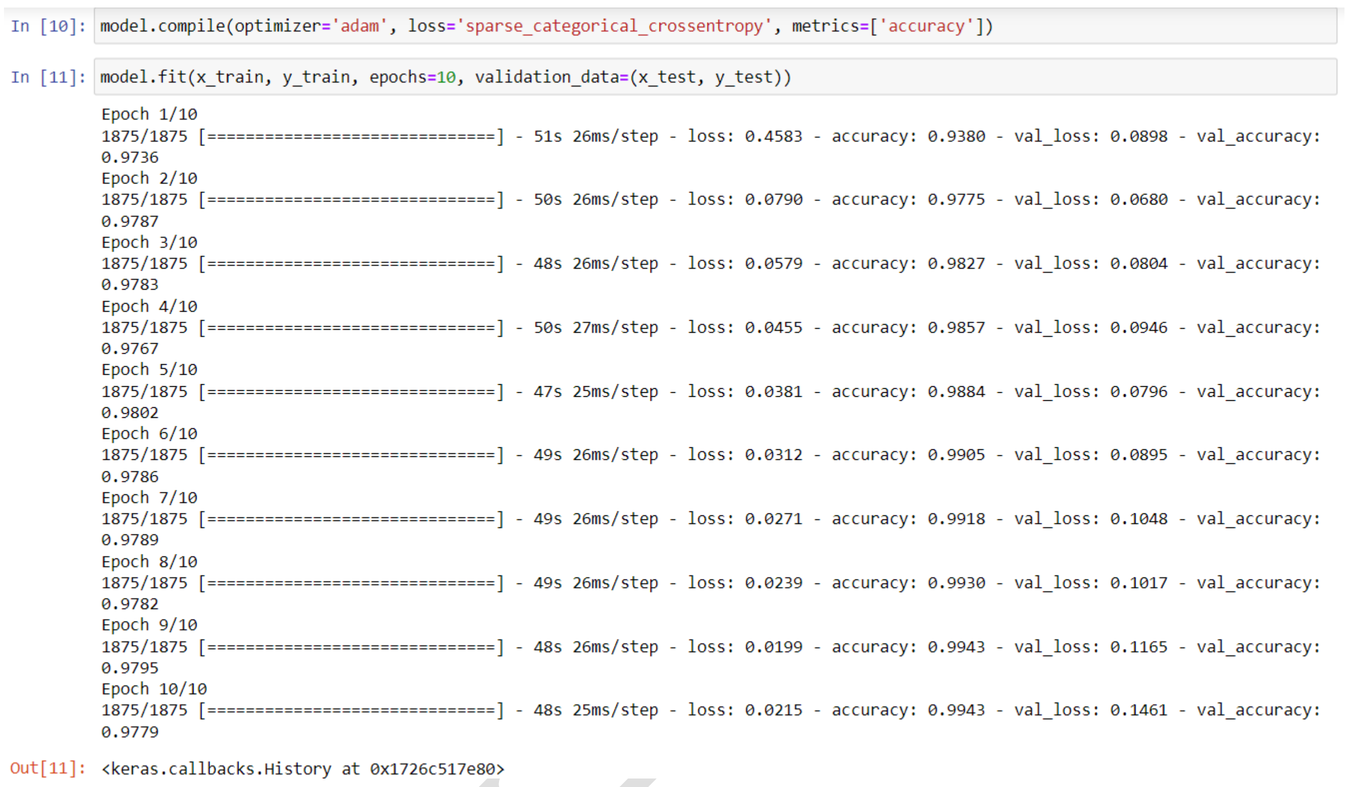


```
In [39]: classes[y_classes[1]]
Out[39]: 'ship'
```

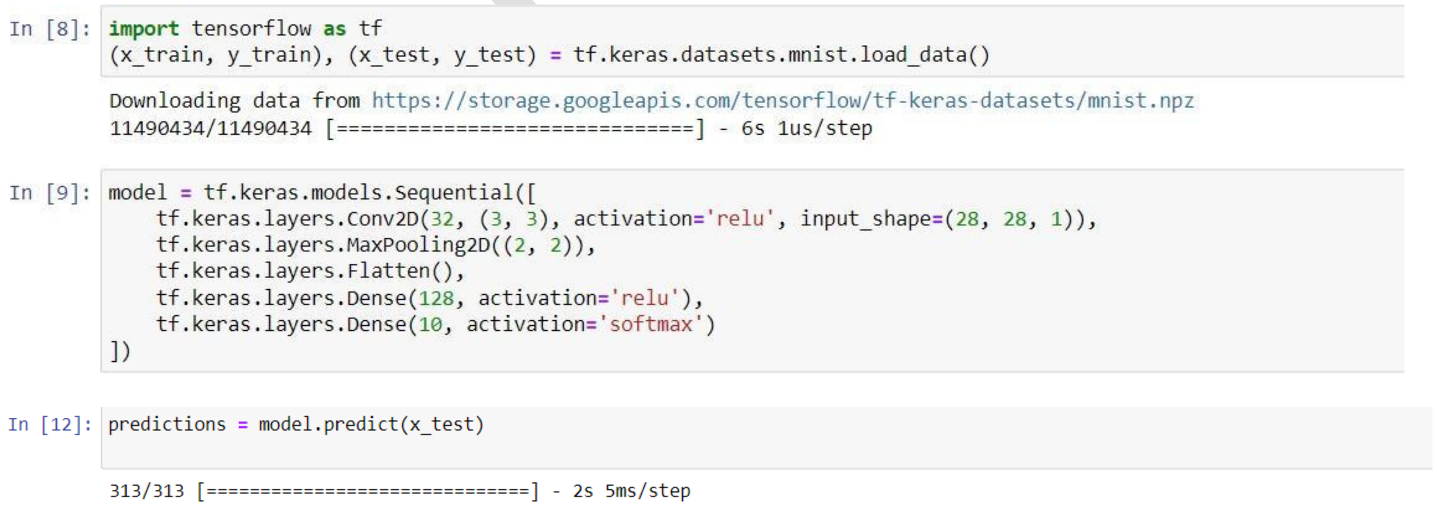
```
In [41]: plot_sample(x_test, y_test,1)
```



## PROGRAM 02 Using MNIST Dataset –



```
In [10]: model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

In [11]: model.fit(x_train, y_train, epochs=10, validation_data=(x_test, y_test))

         Epoch 1/10
         1875/1875 [==============================] - 51s 26ms/step - loss: 0.4583 - accuracy: 0.9380 - val_loss: 0.0898 - val_accuracy:
         0.9736
         Epoch 2/10
         1875/1875 [==============================] - 50s 26ms/step - loss: 0.0790 - accuracy: 0.9775 - val_loss: 0.0680 - val_accuracy:
         0.9787
         Epoch 3/10
         1875/1875 [==============================] - 48s 26ms/step - loss: 0.0579 - accuracy: 0.9827 - val_loss: 0.0804 - val_accuracy:
         0.9783
         Epoch 4/10
         1875/1875 [==============================] - 50s 27ms/step - loss: 0.0455 - accuracy: 0.9857 - val_loss: 0.0946 - val_accuracy:
         0.9767
         Epoch 5/10
         1875/1875 [==============================] - 47s 25ms/step - loss: 0.0381 - accuracy: 0.9884 - val_loss: 0.0796 - val_accuracy:
         0.9802
         Epoch 6/10
         1875/1875 [==============================] - 49s 26ms/step - loss: 0.0312 - accuracy: 0.9905 - val_loss: 0.0895 - val_accuracy:
         0.9786
         Epoch 7/10
         1875/1875 [==============================] - 49s 26ms/step - loss: 0.0271 - accuracy: 0.9918 - val_loss: 0.1048 - val_accuracy:
         0.9789
         Epoch 8/10
         1875/1875 [==============================] - 49s 26ms/step - loss: 0.0239 - accuracy: 0.9930 - val_loss: 0.1017 - val_accuracy:
         0.9782
         Epoch 9/10
         1875/1875 [==============================] - 48s 26ms/step - loss: 0.0199 - accuracy: 0.9943 - val_loss: 0.1165 - val_accuracy:
         0.9795
         Epoch 10/10
         1875/1875 [==============================] - 48s 25ms/step - loss: 0.0215 - accuracy: 0.9943 - val_loss: 0.1461 - val_accuracy:
         0.9779

Out[11]: <keras.callbacks.History at 0x1726c517e80>
```



```
In [8]: import tensorflow as tf
        (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()

        Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
        11490434/11490434 [==============================] - 6s 1us/step

In [9]: model = tf.keras.models.Sequential([
            tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
            tf.keras.layers.MaxPooling2D((2, 2)),
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(128, activation='relu'),
            tf.keras.layers.Dense(10, activation='softmax')
        ])

In [12]: predictions = model.predict(x_test)

         313/313 [==============================] - 2s 5ms/step
```

## Self-Organizing Maps (SOMs):

SOMs, a form of artificial neural network, also known as Kohonen maps, are renowned for their capacity to reduce complexity while retaining the topological characteristics of the data. Data visualization and unsupervised feature extraction may both be done using SOMs.

They can transform high-dimensional data into a low-dimensional representation. SOMs can be helpful for extracting features and data analysis in situations where knowing the data's underlying structure is crucial. SOMs can shed light on data patterns, for instance, in
clustering exercises or exploratory data analysis.

CNNs work well with structured data, such as pictures, but SOMs are frequently used for clustering and visualization across a range of data types. Here's a step-by-step explanation of the SOM algorithm:

Step 1: Initialize the SOM

Step 2: Choose an input data point

Step 3: Compute the Best Matching Unit (neuron with the closest weight vector to the input data point)

Step 4: Update the BMU and its neighbors Step 5: Repeat Steps 2-4

Step 6: Analyse the SOM

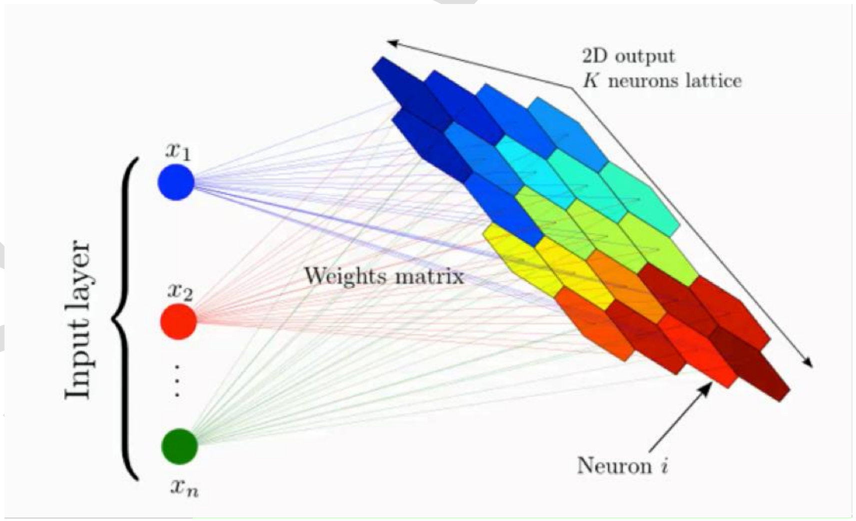
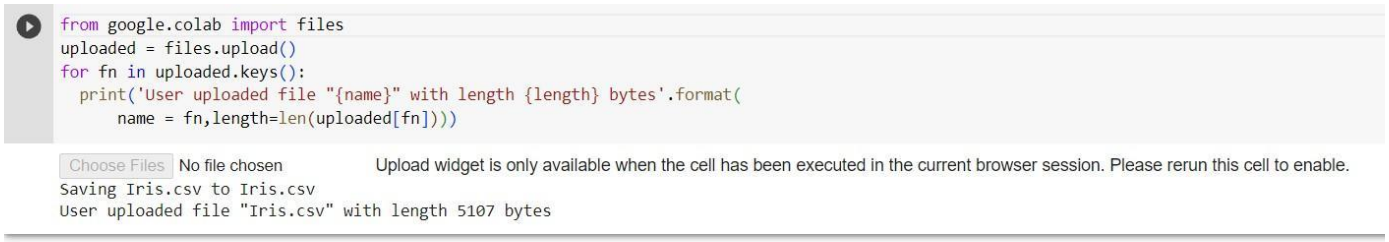


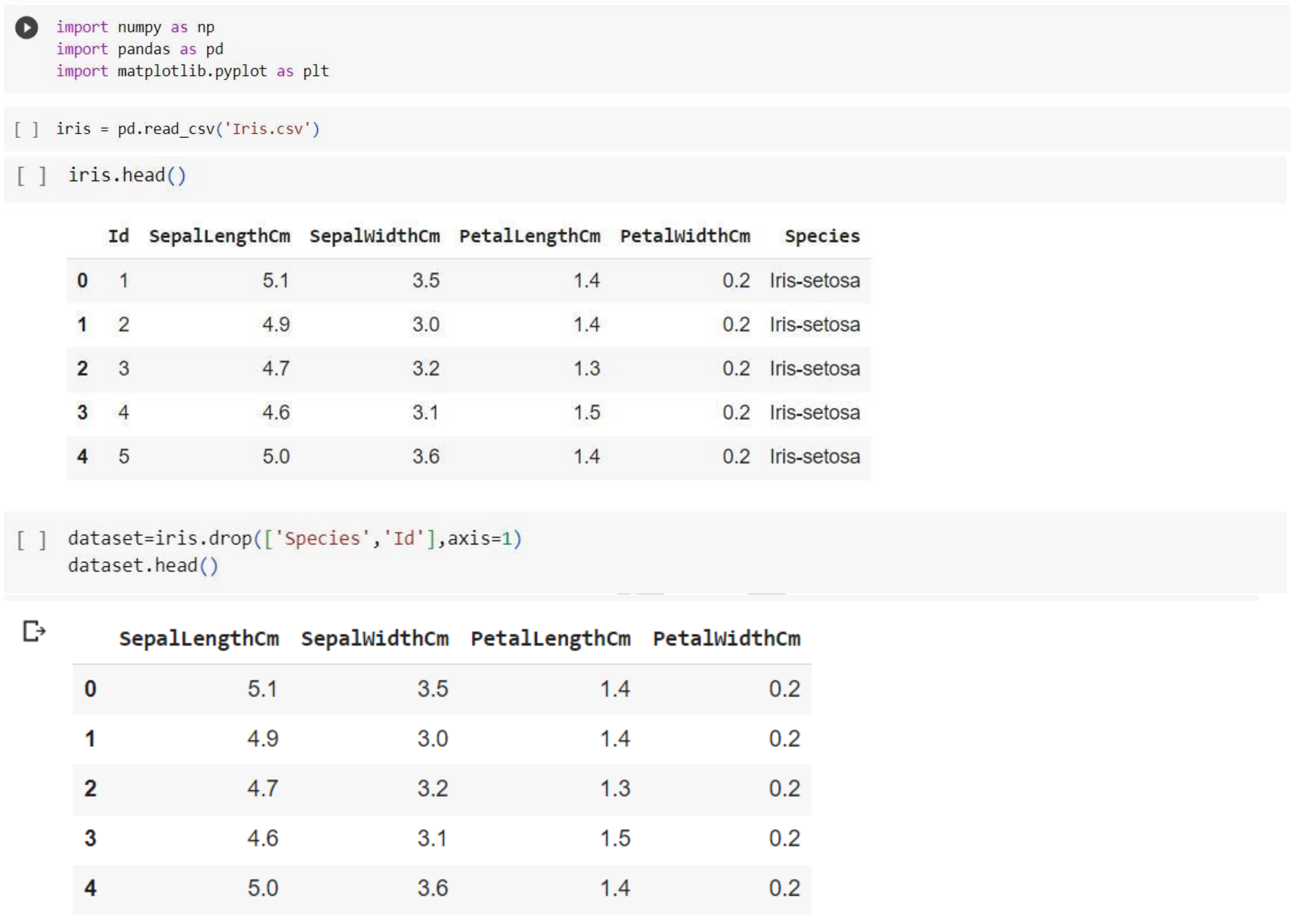
**Fig. 2. Input Layer of CNN**



```python
from google.colab import files
uploaded = files.upload()
for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name = fn,length=len(uploaded[fn])))
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving Iris.csv to Iris.csv
User uploaded file "Iris.csv" with length 5107 bytes
```



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

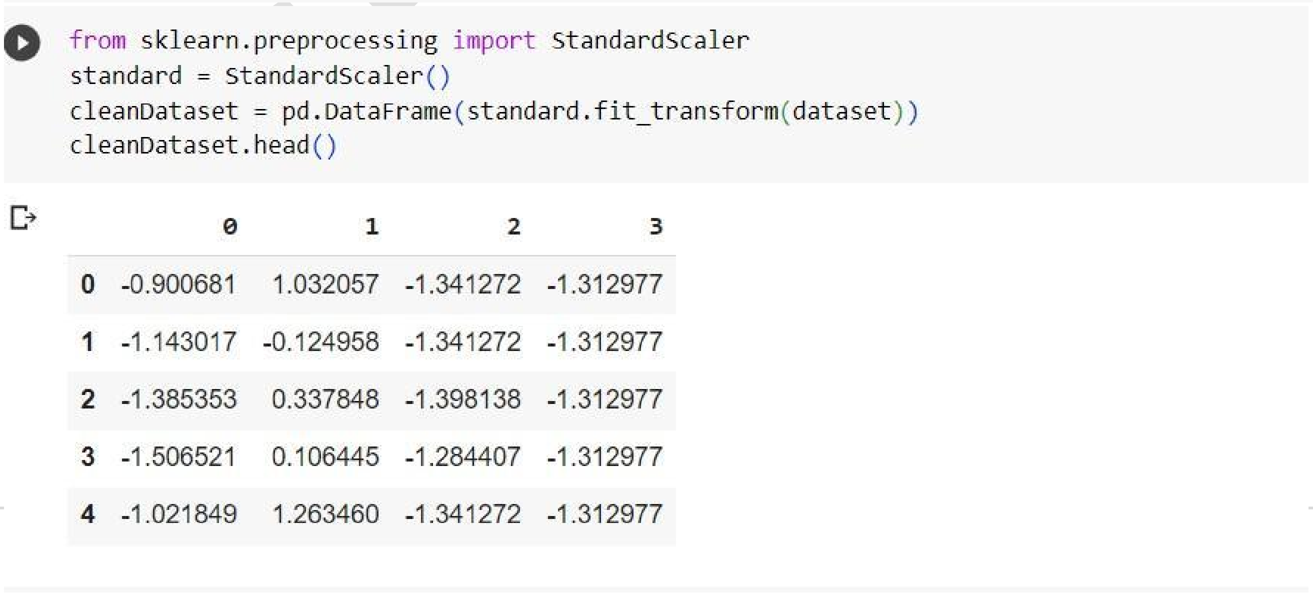
```
[ ] iris = pd.read_csv('Iris.csv')
```

```
[ ] iris.head()
```

| | Id | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm | Species |
|---|---|---|---|---|---|---|
| **0** | 1 | 5.1 | 3.5 | 1.4 | 0.2 | Iris-setosa |
| **1** | 2 | 4.9 | 3.0 | 1.4 | 0.2 | Iris-setosa |
| **2** | 3 | 4.7 | 3.2 | 1.3 | 0.2 | Iris-setosa |
| **3** | 4 | 4.6 | 3.1 | 1.5 | 0.2 | Iris-setosa |
| **4** | 5 | 5.0 | 3.6 | 1.4 | 0.2 | Iris-setosa |

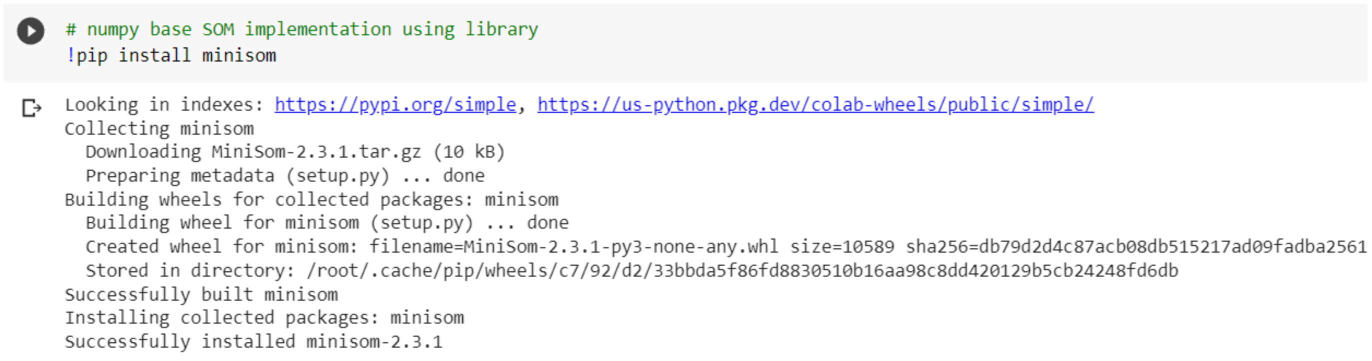
```
[ ] dataset=iris.drop(['Species','Id'],axis=1)
    dataset.head()
```

| | SepalLengthCm | SepalWidthCm | PetalLengthCm | PetalWidthCm |
|---|---|---|---|---|
| **0** | 5.1 | 3.5 | 1.4 | 0.2 |
| **1** | 4.9 | 3.0 | 1.4 | 0.2 |
| **2** | 4.7 | 3.2 | 1.3 | 0.2 |
| **3** | 4.6 | 3.1 | 1.5 | 0.2 |
| **4** | 5.0 | 3.6 | 1.4 | 0.2 |



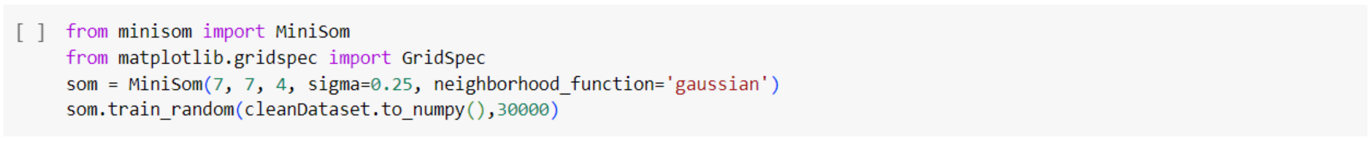
```python
from sklearn.preprocessing import StandardScaler
standard = StandardScaler()
cleanDataset = pd.DataFrame(standard.fit_transform(dataset))
cleanDataset.head()
```

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| **0** | -0.900681 | 1.032057 | -1.341272 | -1.312977 |
| **1** | -1.143017 | -0.124958 | -1.341272 | -1.312977 |
| **2** | -1.385353 | 0.337848 | -1.398138 | -1.312977 |
| **3** | -1.506521 | 0.106445 | -1.284407 | -1.312977 |
| **4** | -1.021849 | 1.263460 | -1.341272 | -1.312977 |

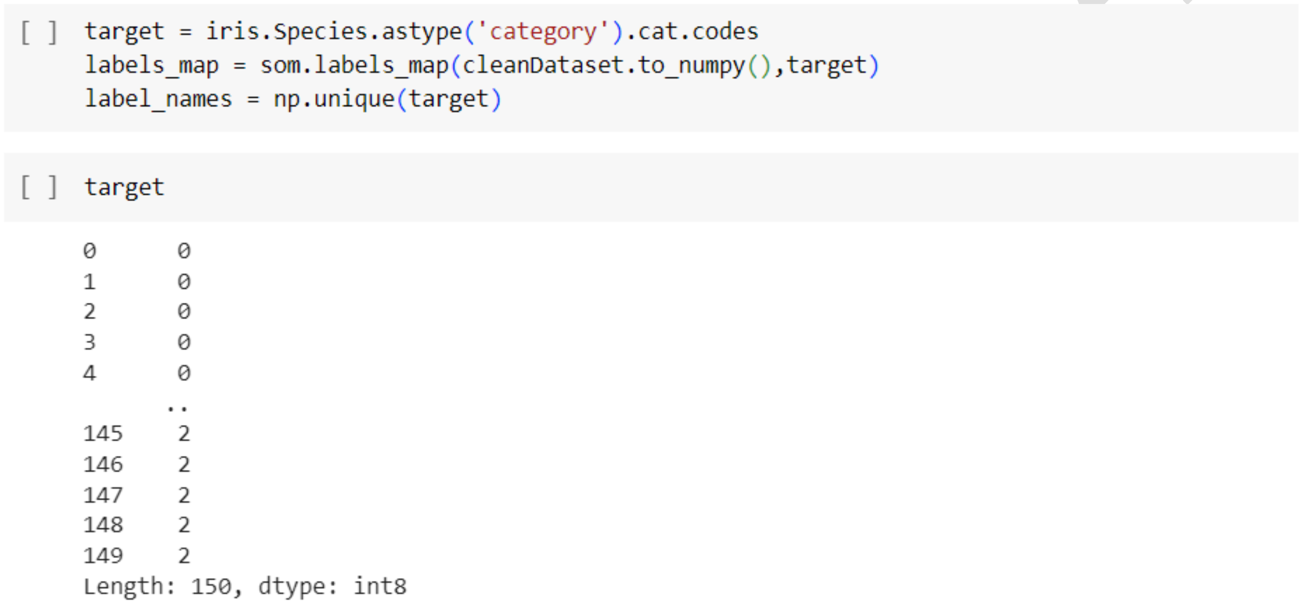


```
# numpy base SOM implementation using library
!pip install minisom
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting minisom
  Downloading MiniSom-2.3.1.tar.gz (10 kB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: minisom
  Building wheel for minisom (setup.py) ... done
  Created wheel for minisom: filename=MiniSom-2.3.1-py3-none-any.whl size=10589 sha256=db79d2d4c87acb08db515217ad09fadba2561
  Stored in directory: /root/.cache/pip/wheels/c7/92/d2/33bbda5f86fd8830510b16aa98c8dd420129b5cb24248fd6db
Successfully built minisom
Installing collected packages: minisom
Successfully installed minisom-2.3.1
```



```
[ ] from minisom import MiniSom
    from matplotlib.gridspec import GridSpec
    som = MiniSom(7, 7, 4, sigma=0.25, neighborhood_function='gaussian')
    som.train_random(cleanDataset.to_numpy(),30000)
```



```
[ ] target = iris.Species.astype('category').cat.codes
    labels_map = som.labels_map(cleanDataset.to_numpy(),target)
    label_names = np.unique(target)
```

```
[ ] target
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Length: 150, dtype: int8
```

```
[ ]  labels_map
```

```
defaultdict(list,
            {(1, 1): Counter({0: 19}),
             (6, 3): Counter({0: 17}),
             (1, 3): Counter({0: 13}),
             (6, 2): Counter({0: 1}),
             (5, 0): Counter({1: 6}),
             (2, 0): Counter({1: 4}),
             (0, 3): Counter({1: 5}),
             (4, 0): Counter({1: 4}),
             (6, 1): Counter({1: 8}),
             (3, 5): Counter({1: 4}),
             (3, 3): Counter({1: 1}),
             (0, 5): Counter({1: 7}),
             (1, 0): Counter({1: 3, 2: 1}),
             (3, 2): Counter({1: 6, 2: 1}),
             (2, 1): Counter({1: 2, 2: 3}),
             (0, 1): Counter({2: 7}),
             (6, 6): Counter({2: 5}),
             (1, 5): Counter({2: 5}),
             (2, 5): Counter({2: 6}),
             (3, 4): Counter({2: 8}),
             (3, 1): Counter({2: 4}),
             (5, 2): Counter({2: 1}),
             (2, 3): Counter({2: 3}),
             (6, 5): Counter({2: 6})})
```

```
[ ]  label_names
```

```
array([0, 1, 2], dtype=int8)
```

```
[ ]  plt.figure(figsize=(7, 7))
     the_grid = GridSpec(7, 7)

     for position in labels_map.keys():
         label_fracs = [labels_map[position][l] for l in label_names]
         plt.subplot(the_grid[6-position[1], position[0]], aspect=1)
         patches, texts = plt.pie(label_fracs)
     plt.legend(patches, label_names, bbox_to_anchor=(0, 1.5), ncol=3)

     plt.show()
```

```python
plt.figure(figsize=(7, 7))
frequencies = np.zeros((7, 7))
for position, values in som.win_map(cleanDataset.to_numpy()).items():
    frequencies[position[0], position[1]] = len(values)
plt.pcolor(frequencies, cmap='Blues')
plt.colorbar()
plt.show()
```

**Optimisation Techniques**

To get the best results, we understood quite a lot of meta-heuristic algorithms, 3 swarm based and 2 math-based algorithms namely Ant Colony Optimisation, Particle Swarm Optimisation, Whale Optimisation, Arithmetic Operation Optimisation and Sine Cosine Algorithm. These algorithms are a class of computational methods used to find high-quality solutions to complex optimization problems. These algorithms are versatile and can be applied to a wide range of problem domains, from engineering and logistics to finance and biology. They are particularly useful when traditional optimization techniques are impractical or inefficient due to the problem's complexity, non-linearity, or high dimensionality.

Metaheuristics are called "meta" because they are higher-level strategies that guide the search for solutions rather than being problem specific. They provide a flexible framework for exploring solution spaces and are often inspired by natural phenomena or human-inspired processes. Some well-known metaheuristic optimization algorithms include Genetic Algorithms (GAs), Particle Swarm Optimization (PSO), Simulated Annealing (SA), Tabu Search (TS), and Ant Colony Optimization (ACO), among others.

We will be discussing about Whale Optimisation, Sine Cosine and Arithmetic Operation Optimization Algorithms.

**Whale Optimisation Algorithm:**

The Whale Optimization Algorithm (WOA) is a nature-inspired metaheuristic optimization algorithm that is based on the social behavior and hunting strategies of humpback whales. It was proposed by Seyedali Mirjalili in 2016 as a relatively new addition to the field of optimization algorithms. WOA is designed to find optimal or near-optimal solutions to a wide range of optimization problems, including continuous, discrete, and combinatorial optimization tasks.

Here's an overview of the key concepts and principles behind the Whale Optimization Algorithm:

a) **Inspiration from Humpback Whales:** The algorithm is inspired by the hunting behavior of humpback whales. Humpback whales are known for their cooperative hunting, where they encircle schools of fish in bubble nets to trap their prey. This cooperative and coordinated behavior serves as the basis for the algorithm's design.

b) **Solution Encoding:** In WOA, potential solutions to the optimization problem are represented as a population of whales. Each whale represents a solution candidate, and the position of a whale corresponds to a potential solution in the search space.

c) **Exploration and Exploitation:** WOA balances exploration and exploitation by employing two main phases: the exploration phase and the exploitation phase. During the exploration phase, the whales move towards a common prey (the global optimum) in a coordinated manner, promoting exploration of the solution space. During the exploitation phase, the whales adjust their positions independently to fine-tune and converge toward the global optimum.

d) **Equations and Formulas:** The algorithm employs several mathematical equations and formulas to model the behavior of whales. These equations include the position update equation, which determines how whales move in the search space, and the encircling prey equation, which models the encircling behavior inspired by real-world whale hunting.

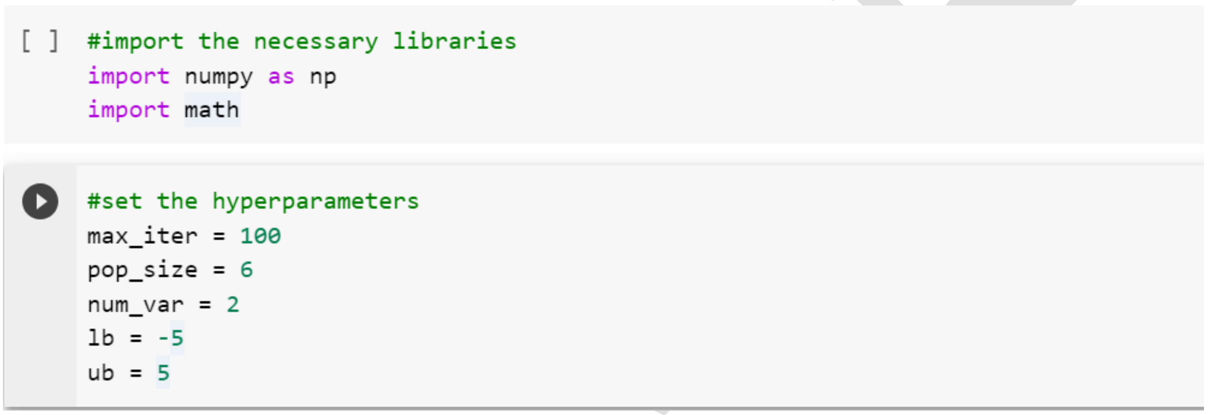
e) **Convergence Mechanism:** The algorithm is designed to converge towards the global optimum by iteratively updating the positions of the whales. It utilizes a diminishing exploration-exploitation factor to gradually shift the focus from exploration to exploitation as the algorithm progresses.

f) **Parameter Tuning:** Like many optimization algorithms, WOA has parameters that require tuning, including the population size, maximum iterations, and control parameters for mathematical equations. Proper parameter tuning is essential for achieving good performance.

g) **Applications:** WOA has been applied to a variety of optimization problems, including function optimization, engineering design, and feature selection in machine learning. It has shown promise in providing competitive results in comparison to other optimization algorithms.

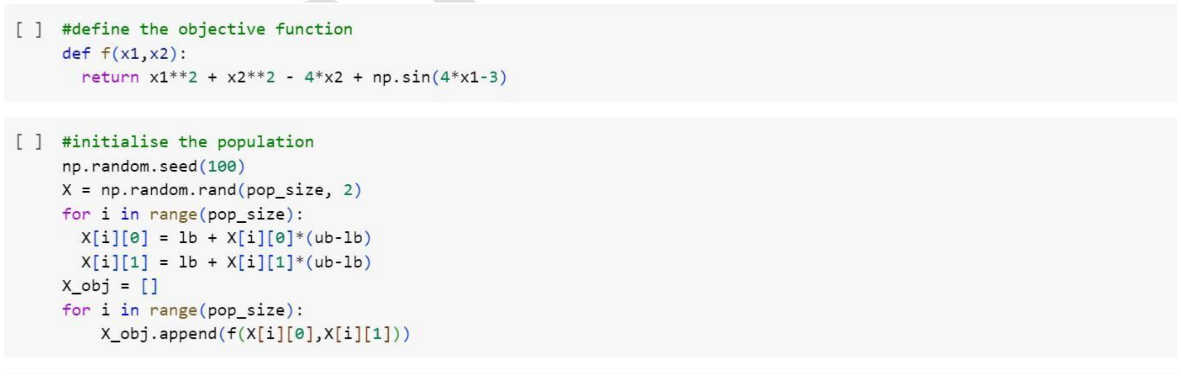
h) **Limitations:** As with any optimization algorithm, WOA may not perform optimally for all types of problems. Its effectiveness can depend on factors like parameter settings, the nature of the problem, and the quality of the initial solutions.

We implemented it in Python by reading the research papers provided by Seyedali Mirjalili. The code for it is as follows:



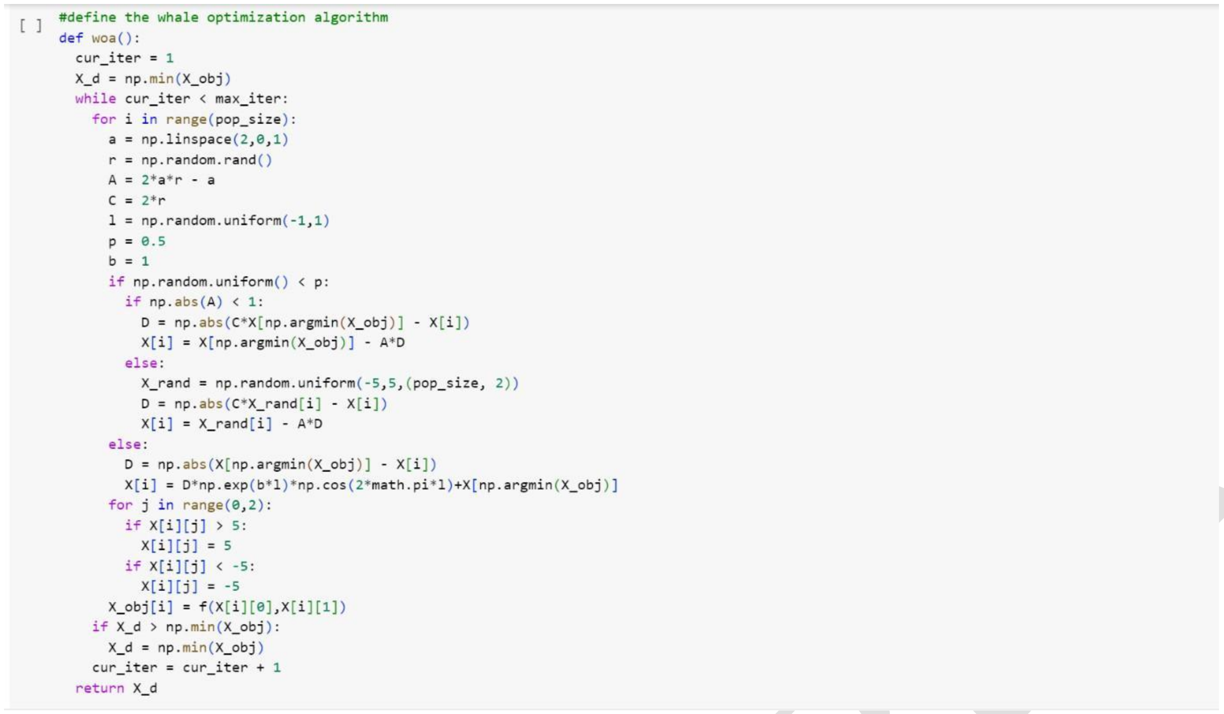
```
[ ]  #import the necessary libraries
     import numpy as np
     import math
```

```
     #set the hyperparameters
     max_iter = 100
     pop_size = 6
     num_var = 2
     lb = -5
     ub = 5
```

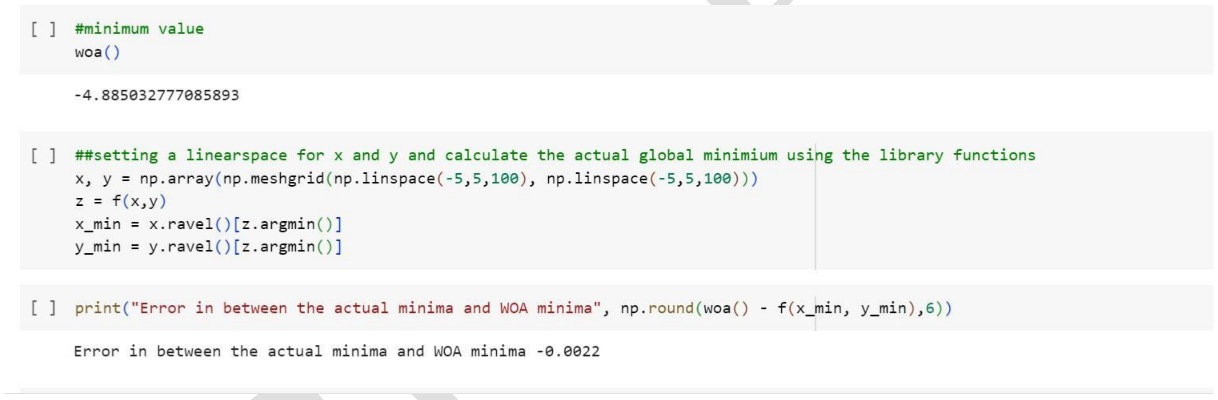


```
[ ]  #define the objective function
     def f(x1,x2):
       return x1**2 + x2**2 - 4*x2 + np.sin(4*x1-3)
```

```
[ ]  #initialise the population
     np.random.seed(100)
     X = np.random.rand(pop_size, 2)
     for i in range(pop_size):
       X[i][0] = lb + X[i][0]*(ub-lb)
       X[i][1] = lb + X[i][1]*(ub-lb)
     X_obj = []
     for i in range(pop_size):
         X_obj.append(f(X[i][0],X[i][1]))
```



```
[ ] #define the whale optimization algorithm
    def woa():
      cur_iter = 1
      X_d = np.min(X_obj)
      while cur_iter < max_iter:
        for i in range(pop_size):
          a = np.linspace(2,0,1)
          r = np.random.rand()
          A = 2*a*r - a
          C = 2*r
          l = np.random.uniform(-1,1)
          p = 0.5
          b = 1
          if np.random.uniform() < p:
            if np.abs(A) < 1:
              D = np.abs(C*X[np.argmin(X_obj)] - X[i])
              X[i] = X[np.argmin(X_obj)] - A*D
            else:
              X_rand = np.random.uniform(-5,5,(pop_size, 2))
              D = np.abs(C*X_rand[i] - X[i])
              X[i] = X_rand[i] - A*D
          else:
            D = np.abs(X[np.argmin(X_obj)] - X[i])
            X[i] = D*np.exp(b*l)*np.cos(2*math.pi*l)+X[np.argmin(X_obj)]
          for j in range(0,2):
            if X[i][j] > 5:
              X[i][j] = 5
            if X[i][j] < -5:
              X[i][j] = -5
          X_obj[i] = f(X[i][0],X[i][1])
        if X_d > np.min(X_obj):
          X_d = np.min(X_obj)
        cur_iter = cur_iter + 1
      return X_d
```



```
[ ] #minimum value
    woa()

    -4.885032777085893
```

```
[ ] ##setting a linearspace for x and y and calculate the actual global minimium using the library functions
    x, y = np.array(np.meshgrid(np.linspace(-5,5,100), np.linspace(-5,5,100)))
    z = f(x,y)
    x_min = x.ravel()[z.argmin()]
    y_min = y.ravel()[z.argmin()]
```

```
[ ] print("Error in between the actual minima and WOA minima", np.round(woa() - f(x_min, y_min),6))

    Error in between the actual minima and WOA minima -0.0022
```

In this, we make use of two techniques namely Bubble net attacking method (exploitation phase) and Search for prey (exploration phase) which is decided upon randomly by a number p which belongs to [0,1]. It is compared with 0.5 because we assume that there is a probability of 50% to choose between either the shrinking encircling mechanism or the spiral model to update the position of whales during optimization.

## Sine Cosine Algorithm:

The Sine Cosine Algorithm (SCA) is a nature-inspired metaheuristic optimization algorithm that was proposed by Seyedali Mirjalili in 2016. This algorithm takes its inspiration from the mathematical properties of sine and cosine functions and aims to efficiently search for optimal or near-optimal solutions to optimization problems, both continuous and discrete.

Here are the key concepts and principles of the Sine Cosine Algorithm:

a) **Mathematical Inspiration:** The SCA is named after the sine and cosine functions, which are fundamental trigonometric functions in mathematics. These functions have periodic behavior and are known for their smooth oscillations.

b) **Solution Representation:** In SCA, potential solutions to an optimization problem are represented as individuals in a population. Each individual is associated with a position in the search space, and the position vector represents a candidate solution.

c) **Initialization:** The algorithm starts by initializing a population of potential solutions randomly within the predefined search space. These initial positions are analogous to the initial values of the sine and cosine functions.

d) **Sine and Cosine Updates:** The key idea behind SCA is the use of sine and cosine functions to adjust the positions of individuals within the population. The sine function is used for exploration, causing the solutions to oscillate around the current positions, while the cosine function is employed for exploitation, leading to convergence towards the best solutions.

e) **Amplitude and Frequency:** SCA introduces two parameters, namely amplitude (A) and frequency (ω), for each individual. The amplitude controls the magnitude of the sine and cosine functions' influence, while the frequency determines how fast the oscillations occur. These parameters are updated iteratively during the optimization process.

f) **Exploration and Exploitation:** SCA strikes a balance between exploration and exploitation. The sine function introduces exploration by causing random oscillations around the current positions, helping the algorithm escape local optima. On the other hand, the cosine function promotes exploitation by guiding the search towards promising regions of the search space.

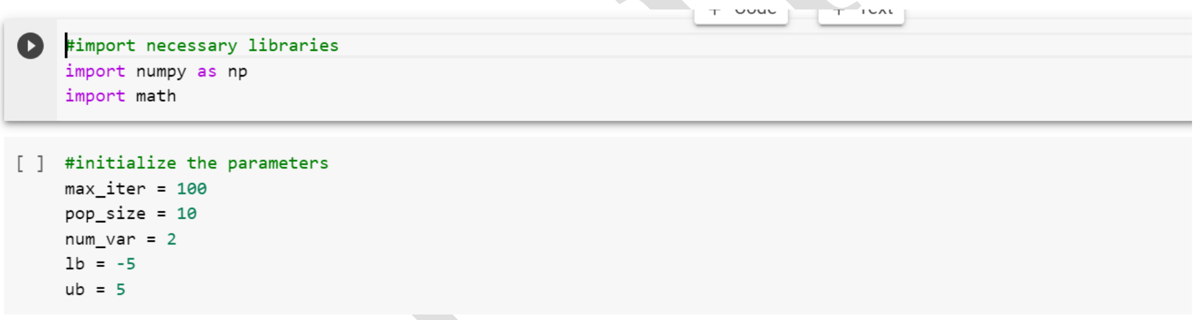
g) **Termination Criterion:** Like other optimization algorithms, SCA typically includes a termination criterion to stop the search process. This could be based on a maximum number of iterations, a tolerance threshold, or other user- defined criteria.

h) **Applications:** SCA has been applied to a wide range of optimization problems, including function optimization, parameter tuning in machine learning algorithms, feature selection, and engineering design. Its versatility makes it suitable for various domains.

i) **Parameter Tuning:** As with many optimization algorithms, SCA involves parameter tuning, such as setting the population size, the maximum number of iterations, and the control parameters for the sine and cosine functions. Proper parameter tuning is essential for achieving good performance.

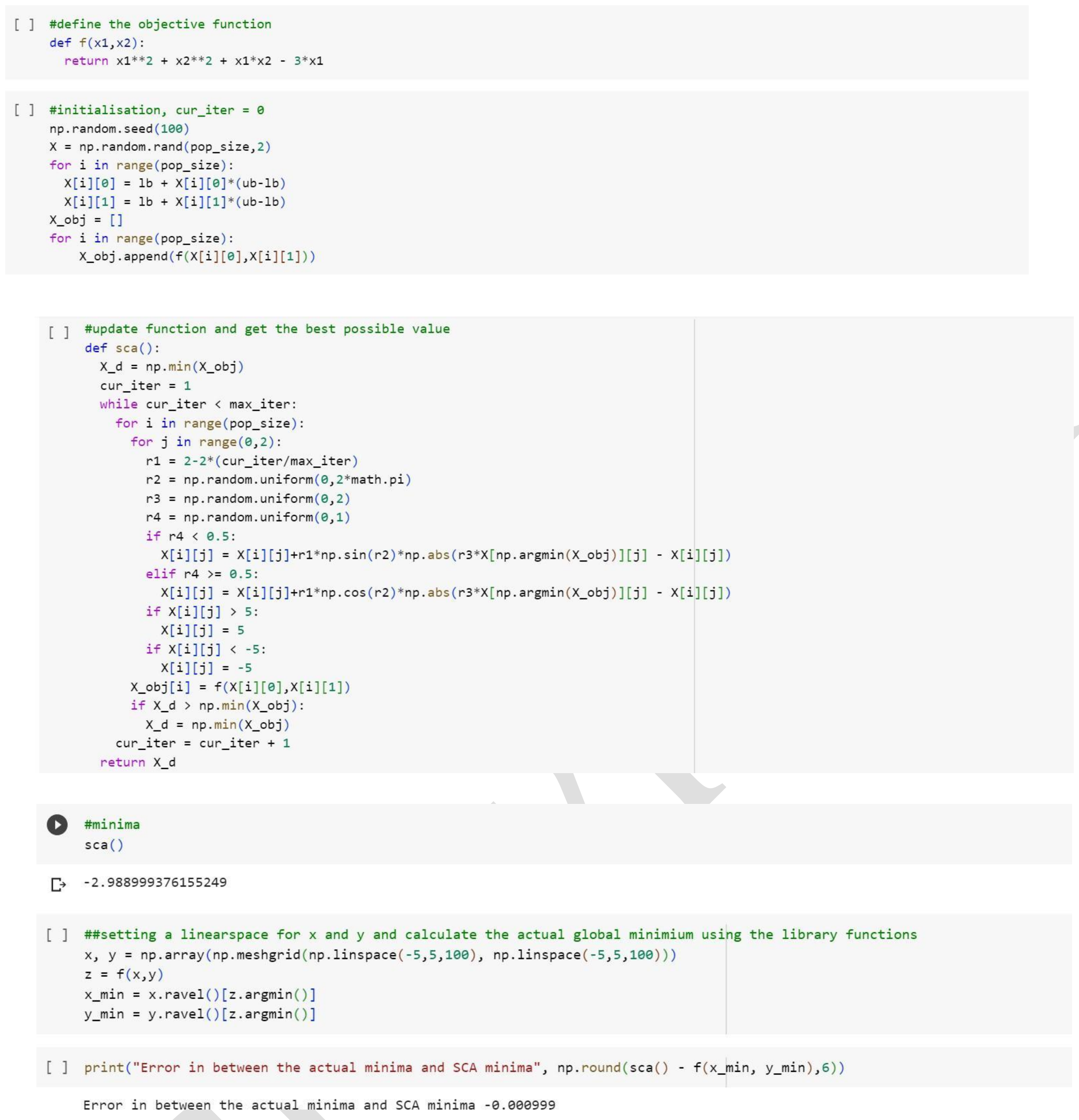
j) **Performance:** SCA has shown promise in providing competitive results when compared to other optimization algorithms. Its performance can vary depending on the problem type, the quality of the initial solutions, and parameter settings.

We implemented it in Python by reading the research papers provided by Seyedali Mirjalili. The code for it is as follows:



```
#import necessary libraries
import numpy as np
import math
```

```
#initialize the parameters
max_iter = 100
pop_size = 10
num_var = 2
lb = -5
ub = 5
```



```
[ ] #define the objective function
    def f(x1,x2):
      return x1**2 + x2**2 + x1*x2 - 3*x1
```

```
[ ] #initialisation, cur_iter = 0
    np.random.seed(100)
    X = np.random.rand(pop_size,2)
    for i in range(pop_size):
      X[i][0] = lb + X[i][0]*(ub-lb)
      X[i][1] = lb + X[i][1]*(ub-lb)
    X_obj = []
    for i in range(pop_size):
        X_obj.append(f(X[i][0],X[i][1]))
```

```
[ ] #update function and get the best possible value
    def sca():
      X_d = np.min(X_obj)
      cur_iter = 1
      while cur_iter < max_iter:
        for i in range(pop_size):
          for j in range(0,2):
            r1 = 2-2*(cur_iter/max_iter)
            r2 = np.random.uniform(0,2*math.pi)
            r3 = np.random.uniform(0,2)
            r4 = np.random.uniform(0,1)
            if r4 < 0.5:
              X[i][j] = X[i][j]+r1*np.sin(r2)*np.abs(r3*X[np.argmin(X_obj)][j] - X[i][j])
            elif r4 >= 0.5:
              X[i][j] = X[i][j]+r1*np.cos(r2)*np.abs(r3*X[np.argmin(X_obj)][j] - X[i][j])
            if X[i][j] > 5:
              X[i][j] = 5
            if X[i][j] < -5:
              X[i][j] = -5
          X_obj[i] = f(X[i][0],X[i][1])
          if X_d > np.min(X_obj):
            X_d = np.min(X_obj)
        cur_iter = cur_iter + 1
      return X_d
```

```
#minima
sca()

-2.988999376155249
```

```
[ ] ##setting a linearspace for x and y and calculate the actual global minimium using the library functions
    x, y = np.array(np.meshgrid(np.linspace(-5,5,100), np.linspace(-5,5,100)))
    z = f(x,y)
    x_min = x.ravel()[z.argmin()]
    y_min = y.ravel()[z.argmin()]
```

```
[ ] print("Error in between the actual minima and SCA minima", np.round(sca() - f(x_min, y_min),6))

    Error in between the actual minima and SCA minima -0.000999
```

We are making use of the trigonometric and periodic properties of sine and cosine which helps us exploit and surface the search space well. Also, the range for both of these functions range from -1 to 1 which in turn helps us scan the regions locally to find the global maxima or minima. Description of the control parameters is as follows:

r1 – Helps regulate exploration and exploitation phases in early and later stages r2 – Helps regulate how far the search agents deviate from the optimal point

r3 – Random scaling factor used for regulating the movement

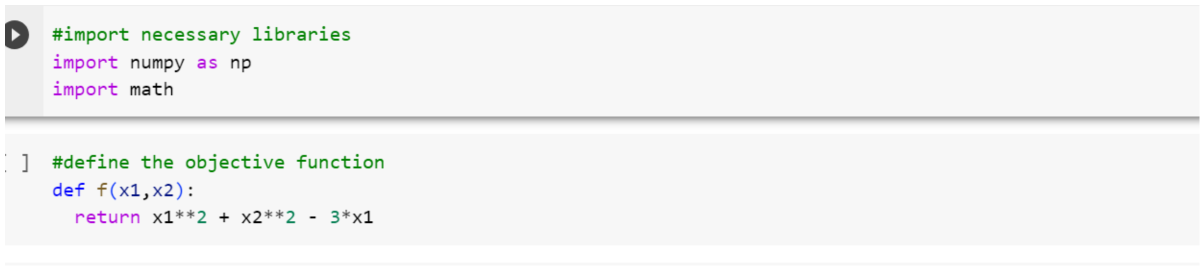
r4 – Randomly switch between sine and cosine components in position update equations

**_Arithmetic Optimization Algorithm**

Abualigah and colleagues introduced an innovative population-based metaheuristic method known as the Arithmetic Optimization Algorithm (AOA). This approach leverages the statistical distribution characteristics of the fundamental arithmetic operators in mathematics, which include Multiplication (M"×"), Division (D"÷"), Subtraction (S"−"), and Addition (A"+"). Directly quoting the research paper regarding how it was inspired:

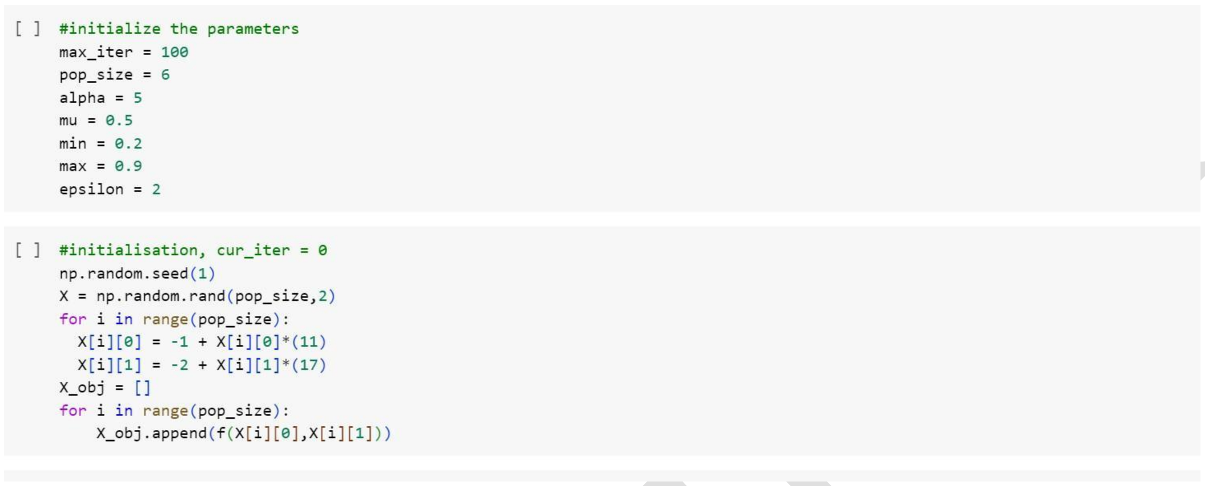
“Arithmetic is a fundamental component of number theory, and it is one of the important parts of modern mathematics, along with geometry, algebra, and analysis. Arithmetic operators (i.e., Multiplication, Division, Subtraction, and Addition) are the traditional calculation measures used usually to study the numbers. We use these simple operators as a mathematical optimization to determine the best element subjected to specific criteria from some set of candidate alternatives (solutions). Optimization problems occur in all quantitative disciplines from engineering, economics, and computer sciences to operations research and industry, and the improvement of solution techniques has attracted the interest of mathematics for eras.

The main inspiration of the proposed AOA arises from the use of Arithmetic operators in solving the Arithmetic problems. “



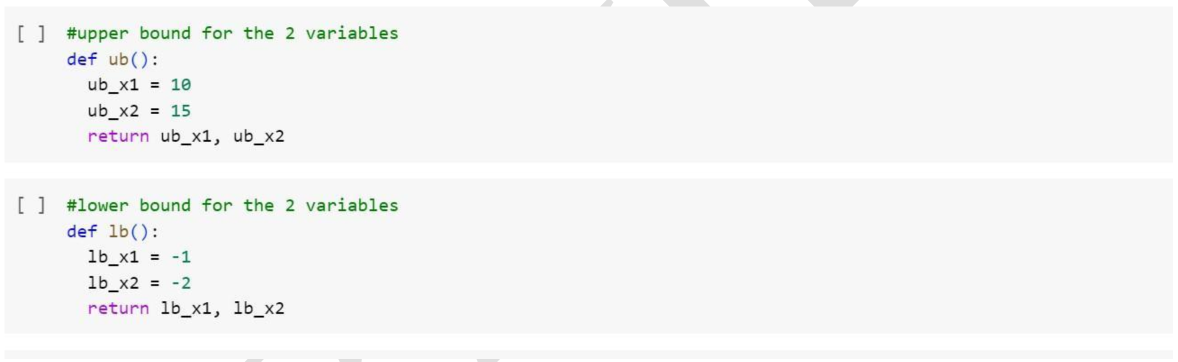
```python
#import necessary libraries
import numpy as np
import math
```

```python
#define the objective function
def f(x1,x2):
  return x1**2 + x2**2 - 3*x1
```



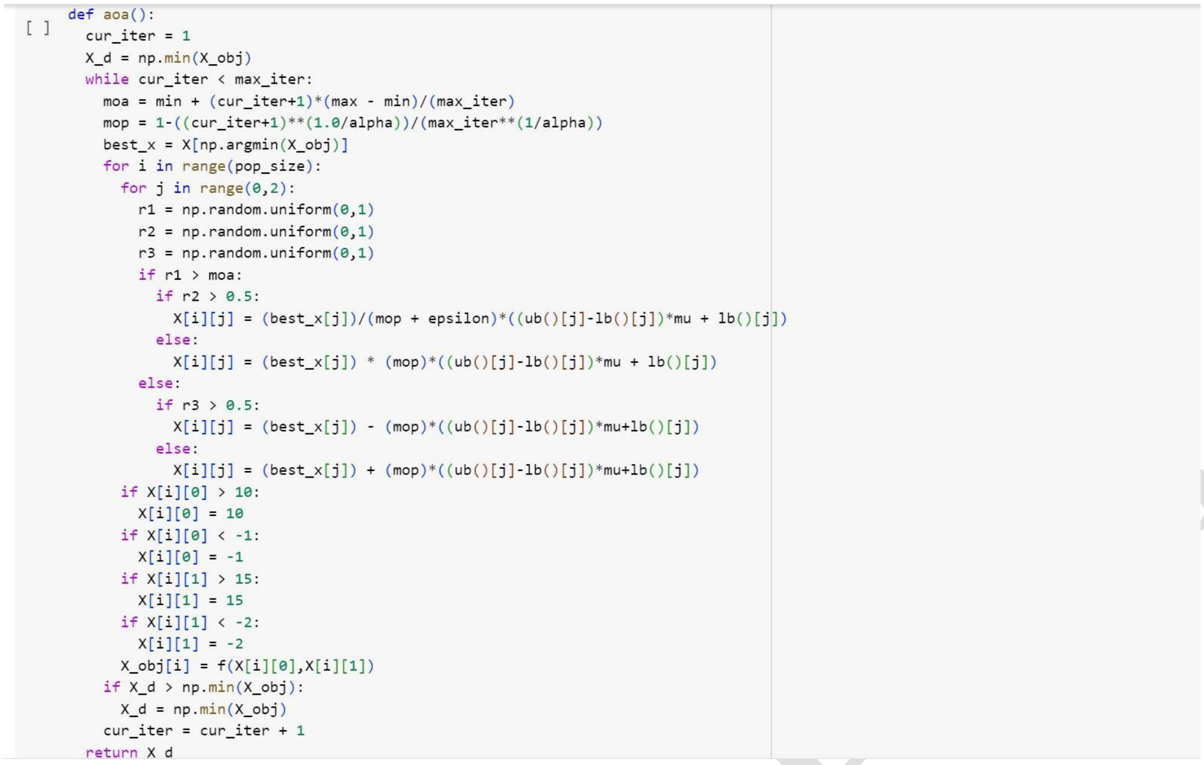
```python
#initialize the parameters
max_iter = 100
pop_size = 6
alpha = 5
mu = 0.5
min = 0.2
max = 0.9
epsilon = 2
```

```python
#initialisation, cur_iter = 0
np.random.seed(1)
X = np.random.rand(pop_size,2)
for i in range(pop_size):
  X[i][0] = -1 + X[i][0]*(11)
  X[i][1] = -2 + X[i][1]*(17)
X_obj = []
for i in range(pop_size):
    X_obj.append(f(X[i][0],X[i][1]))
```

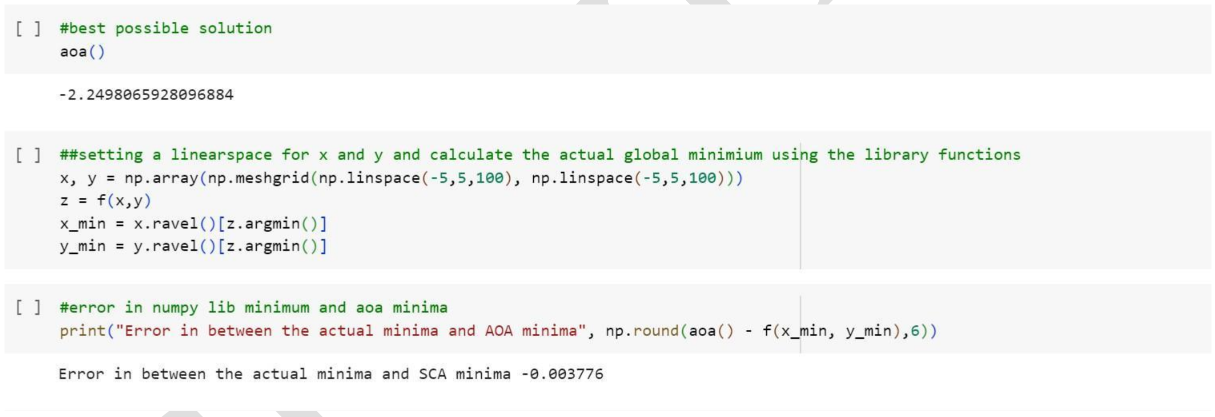


```python
#upper bound for the 2 variables
def ub():
  ub_x1 = 10
  ub_x2 = 15
  return ub_x1, ub_x2
```

```python
#lower bound for the 2 variables
def lb():
  lb_x1 = -1
  lb_x2 = -2
  return lb_x1, lb_x2
```



```
[ ] def aoa():
      cur_iter = 1
      X_d = np.min(X_obj)
      while cur_iter < max_iter:
        moa = min + (cur_iter+1)*(max - min)/(max_iter)
        mop = 1-((cur_iter+1)**(1.0/alpha))/(max_iter**(1/alpha))
        best_x = X[np.argmin(X_obj)]
        for i in range(pop_size):
          for j in range(0,2):
            r1 = np.random.uniform(0,1)
            r2 = np.random.uniform(0,1)
            r3 = np.random.uniform(0,1)
            if r1 > moa:
              if r2 > 0.5:
                X[i][j] = (best_x[j])/(mop + epsilon)*((ub()[j]-lb()[j])*mu + lb()[j])
              else:
                X[i][j] = (best_x[j]) * (mop)*((ub()[j]-lb()[j])*mu + lb()[j])
            else:
              if r3 > 0.5:
                X[i][j] = (best_x[j]) - (mop)*((ub()[j]-lb()[j])*mu+lb()[j])
              else:
                X[i][j] = (best_x[j]) + (mop)*((ub()[j]-lb()[j])*mu+lb()[j])
          if X[i][0] > 10:
            X[i][0] = 10
          if X[i][0] < -1:
            X[i][0] = -1
          if X[i][1] > 15:
            X[i][1] = 15
          if X[i][1] < -2:
            X[i][1] = -2
          X_obj[i] = f(X[i][0],X[i][1])
        if X_d > np.min(X_obj):
          X_d = np.min(X_obj)
        cur_iter = cur_iter + 1
      return X_d
```



```
[ ] #best possible solution
    aoa()

    -2.2498065928096884
```

```
[ ] ##setting a linearspace for x and y and calculate the actual global minimium using the library functions
    x, y = np.array(np.meshgrid(np.linspace(-5,5,100), np.linspace(-5,5,100)))
    z = f(x,y)
    x_min = x.ravel()[z.argmin()]
    y_min = y.ravel()[z.argmin()]
```

```
[ ] #error in numpy lib minimum and aoa minima
    print("Error in between the actual minima and AOA minima", np.round(aoa() - f(x_min, y_min),6))

    Error in between the actual minima and SCA minima -0.003776
```

In this, we make use of the arithmetic operators Addition (+), Subtraction (-), Multiplication (x) and Division (÷) which help us undergo Exploration (diversification) and Exploitation (intensification) stages of the meta heuristic algorithms. We make use of division and multiplication (due to their property of giving us high distributed values) for exploration and addition, subtraction for exploitation because these operators will give us less dispersed values and hence help us reach the required optima. We have calculated a MOA (Math Optimizer Accelerated) which act as a coefficient in order to find the search phase, whether to go for exploration or exploitation. As per the paper's findings some values such as α and μ were fixed as they proved to give the most accurate results as per the experiments conducted by them. α and μ are exploitation accuracy and control parameter to adjust search phases respectively.

## Classification Algorithms

### Support Vector Machine

SVMs are a strong supervised machine learning technique that is used for regression and classification tasks. They are very useful for two-class classification jobs. The primary principle underlying SVM is to find a hyperplane that best separates data points from distinct classes while maximising the margin between them. The hyperplane chosen maximises the distance between the nearest data points (referred to as support vectors) from each class.

Here's a quick overview of SVM and an illustration of Python code snippet utilizing the widely used Scikit-Learn library:

***WORKING***

_**Data Preparation**: You begin by creating your labelled dataset, which represents each data point as a feature vector with accompanying class labels.

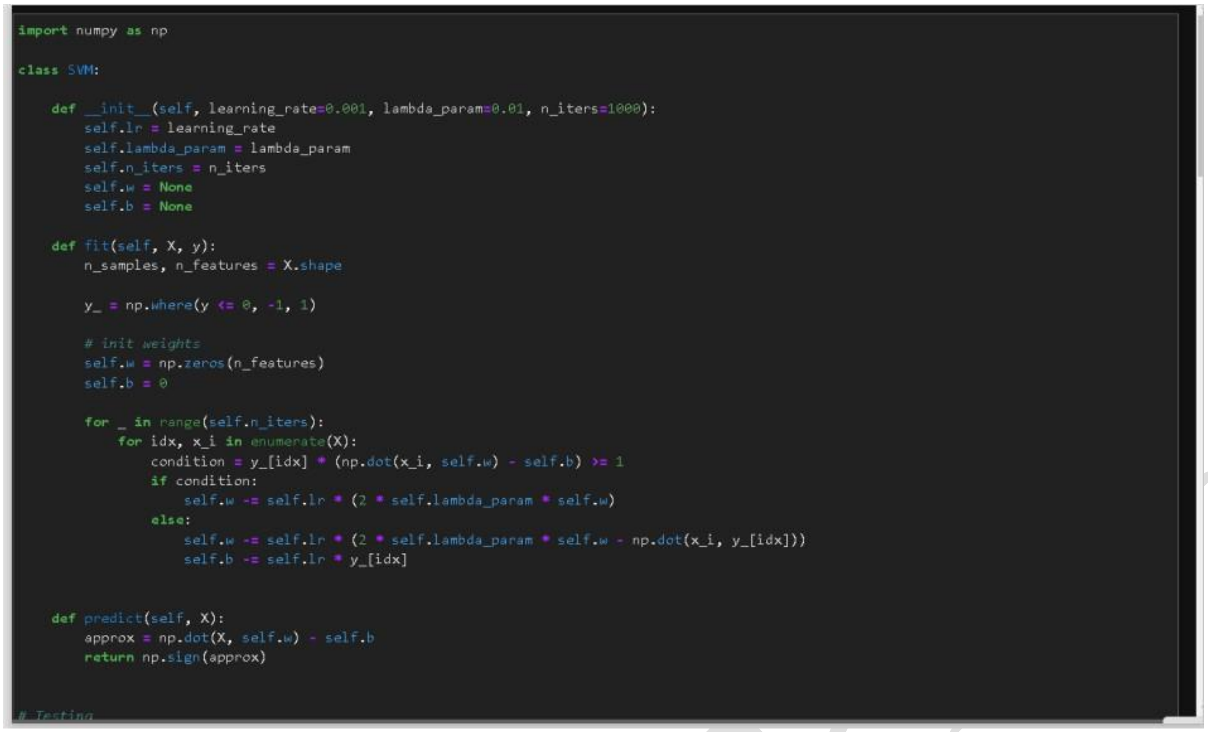
**Feature Scaling:** Because SVM is susceptible to the size of the input features, it is sometimes necessary to scale or normalise the characteristics so that they have similar ranges.

SVM can employ a variety of kernel functions (linear, polynomial, radial basis function, etc.) to transform data into a space with additional dimensions. The kernel you use is determined by the nature of your data and the problem you're attempting to solve.

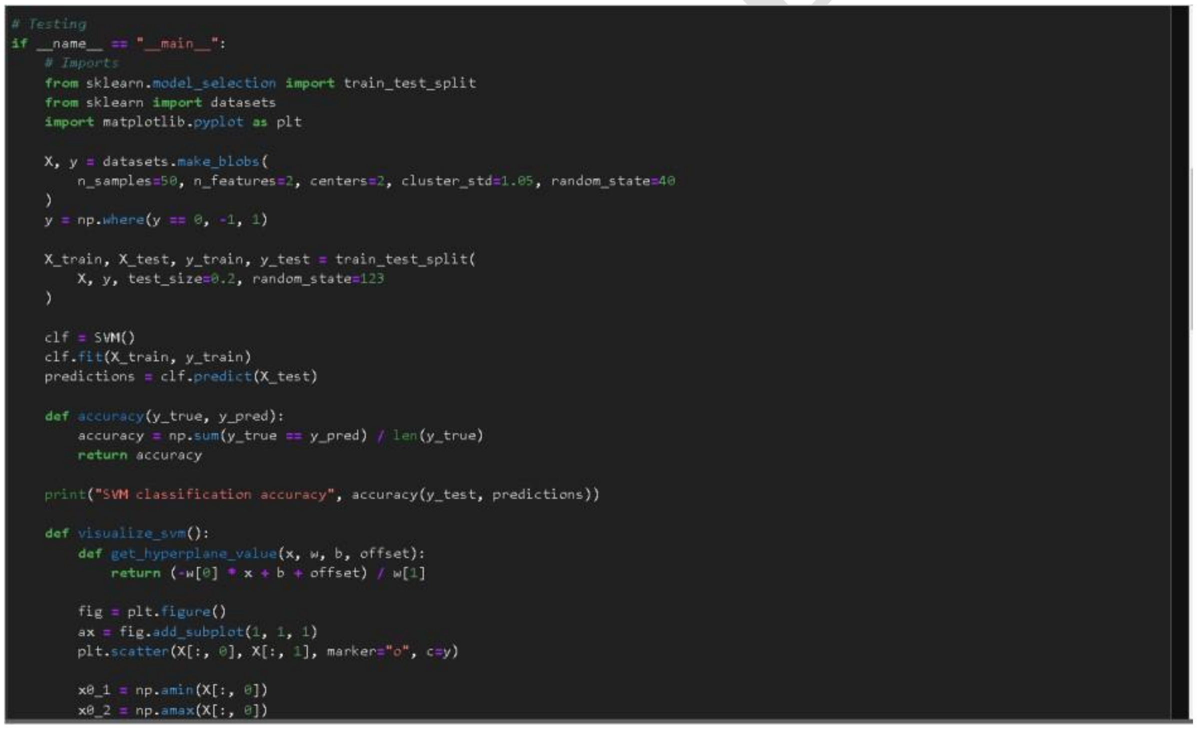
**Training:** The SVM method seeks the hyperplane with the greatest margin between classes while minimising classification errors. This is usually accomplished using optimisation approaches such as the Sequential Minimal Optimisation (SMO) algorithm. SVM identifies the support vectors, which are the data points nearest to the decision boundary, during training.

**Prediction:** After training, the SVM model can be used to forecast the class labels of new, previously unseen data points by evaluating which side of the hyperplane they fall on.

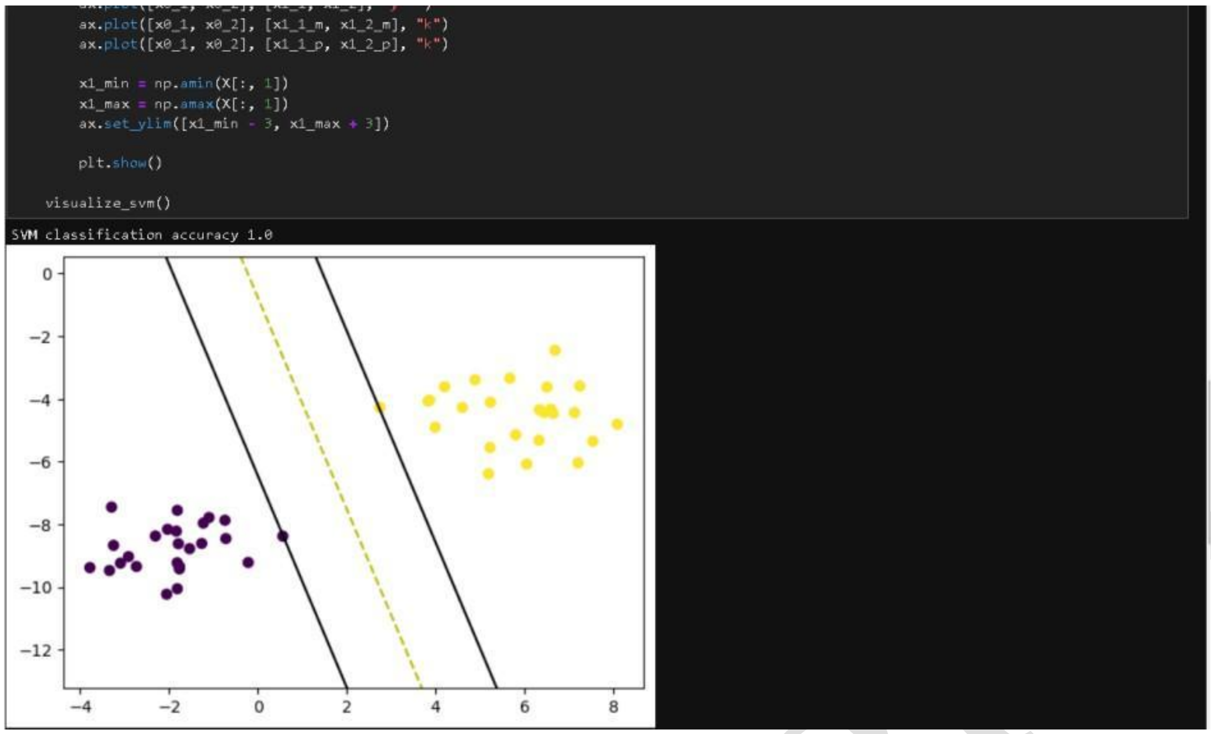
## CODE



```python
import numpy as np

class SVM:

    def __init__(self, learning_rate=0.001, lambda_param=0.01, n_iters=1000):
        self.lr = learning_rate
        self.lambda_param = lambda_param
        self.n_iters = n_iters
        self.w = None
        self.b = None

    def fit(self, X, y):
        n_samples, n_features = X.shape

        y_ = np.where(y <= 0, -1, 1)

        # init weights
        self.w = np.zeros(n_features)
        self.b = 0

        for _ in range(self.n_iters):
            for idx, x_i in enumerate(X):
                condition = y_[idx] * (np.dot(x_i, self.w) - self.b) >= 1
                if condition:
                    self.w -= self.lr * (2 * self.lambda_param * self.w)
                else:
                    self.w -= self.lr * (2 * self.lambda_param * self.w - np.dot(x_i, y_[idx]))
                    self.b -= self.lr * y_[idx]


    def predict(self, X):
        approx = np.dot(X, self.w) - self.b
        return np.sign(approx)


# Testing
```



```python
# Testing
if __name__ == "__main__":
    # Imports
    from sklearn.model_selection import train_test_split
    from sklearn import datasets
    import matplotlib.pyplot as plt

    X, y = datasets.make_blobs(
        n_samples=50, n_features=2, centers=2, cluster_std=1.05, random_state=40
    )
    y = np.where(y == 0, -1, 1)

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=123
    )

    clf = SVM()
    clf.fit(X_train, y_train)
    predictions = clf.predict(X_test)

    def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy

    print("SVM classification accuracy", accuracy(y_test, predictions))

    def visualize_svm():
        def get_hyperplane_value(x, w, b, offset):
            return (-w[0] * x + b + offset) / w[1]

        fig = plt.figure()
        ax = fig.add_subplot(1, 1, 1)
        plt.scatter(X[:, 0], X[:, 1], marker="o", c=y)

        x0_1 = np.amin(X[:, 0])
        x0_2 = np.amax(X[:, 0])
```



```
        ax.plot([x0_1, x0_2], [x1_1_m, x1_2_m], "k")
        ax.plot([x0_1, x0_2], [x1_1_p, x1_2_p], "k")

        x1_min = np.amin(X[:, 1])
        x1_max = np.amax(X[:, 1])
        ax.set_ylim([x1_min - 3, x1_max + 3])

        plt.show()

    visualize_svm()
```

```
SVM classification accuracy 1.0
```

**SUMMARY**

- We load a sample dataset and specify its features.
- Split it into training and testing sets.
- Create an SVM classifier with a linear kernel.
- Train the classifier on the training data.
- Make predictions on the test data.

Evaluate the accuracy of the classifier on the test *data* which comes out to be 1.00

**Decision Trees**

A Decision Tree is a supervised machine learning technique that can be used for
classification and regression. It is a tree-like architecture in which every internal node represents a feature , each branch represents a decision rule, and each leaf node represents a result (or class label or value). Decision trees are well-known for their clarity and are frequently utilised in a variety of industries such as business, medical, and finance due to their capacity to provide details about methods of decision-making.

**Working**

_Data Preparation: Begin with a labelled dataset in which every point of data has a collection of features and a corresponding target variable.

Choosing the Splitting Criteria: The procedure chooses a feature and a value to divide the data into subsets (child nodes) that result in the most homogeneous groups in terms of the target variable.

Making the Tree: The process of creating trees is recursive. It begins at the root node and repeatedly selects the best feature to partition the data, resulting in the creation of child. nodes. This process is repeated until a stopping requirement, such as a maximum depth or a minimum number of samples per leaf, is fulfilled.

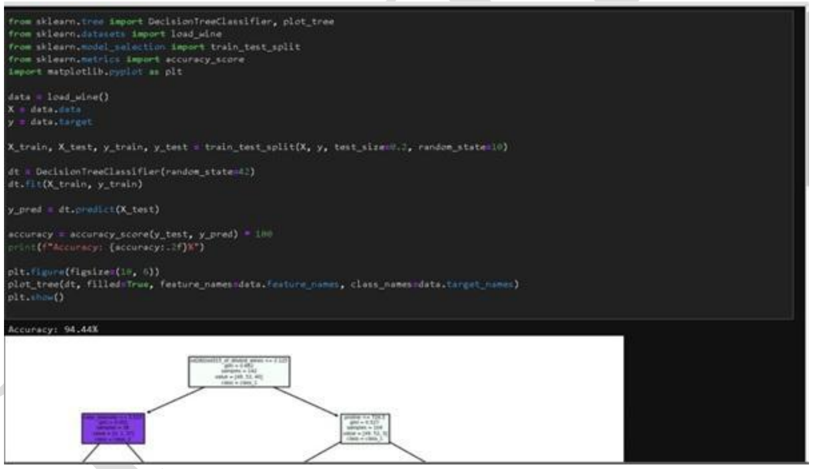
**Code**

- We load the load_wine dataset.
- Split it into training and testing sets.
- Create a Decision Tree classifier.
- Train the classifier on the training data.
- Make predictions on the test data and plot the decision tree.
- Evaluate the accuracy of the classifier on the test data which comes out to be 94.44%

**SUMMARY**

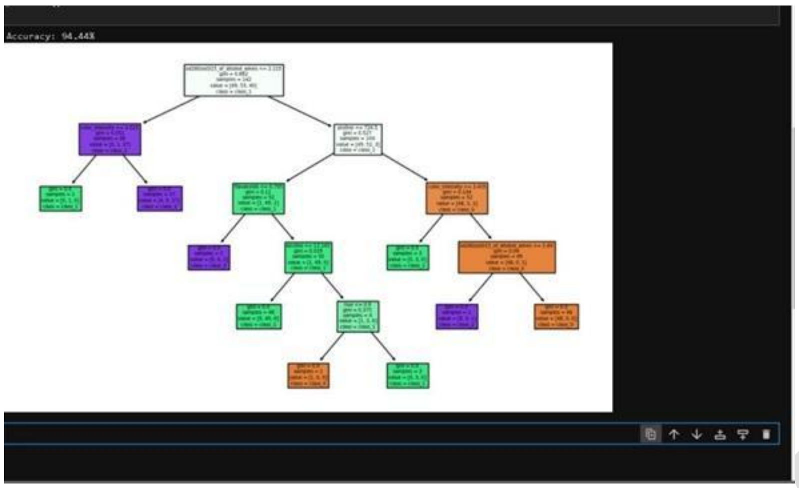
**KNN**

The k-Nearest Neighbours (k-NN) technique is a straightforward supervised machine learning approach that may be used for regression, classification, and other applications. It works on the premise of locating the k-nearest data points (neighbours) in the training dataset to a new, unknown data point and making projections based on the majority class (for classification) the closest neighbours.



```
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

data = load_wine()
X = data.data
y = data.target

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10)

dt = DecisionTreeClassifier(random_state=42)
dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

accuracy = accuracy_score(y_test, y_pred) * 100
print(f"Accuracy: {accuracy:.2f}%")

plt.figure(figsize=(10, 6))
plot_tree(dt, filled=True, feature_names=data.feature_names, class_names=data.target_names)
plt.show()
```

```
Accuracy: 94.44%
```



```
Accuracy: 94.44%
```

**WORKING**

k-NN predicts a new data point's class label by locating its k-nearest neighbours and selecting the majority class from among them.

It predicts a numerical value for regression by averaging the target values of the k-nearest neighbours.

The selection of parameters and the scale of features are critical. It is straightforward and easy to understand, although it may struggle with huge or high-dimensional datasets.

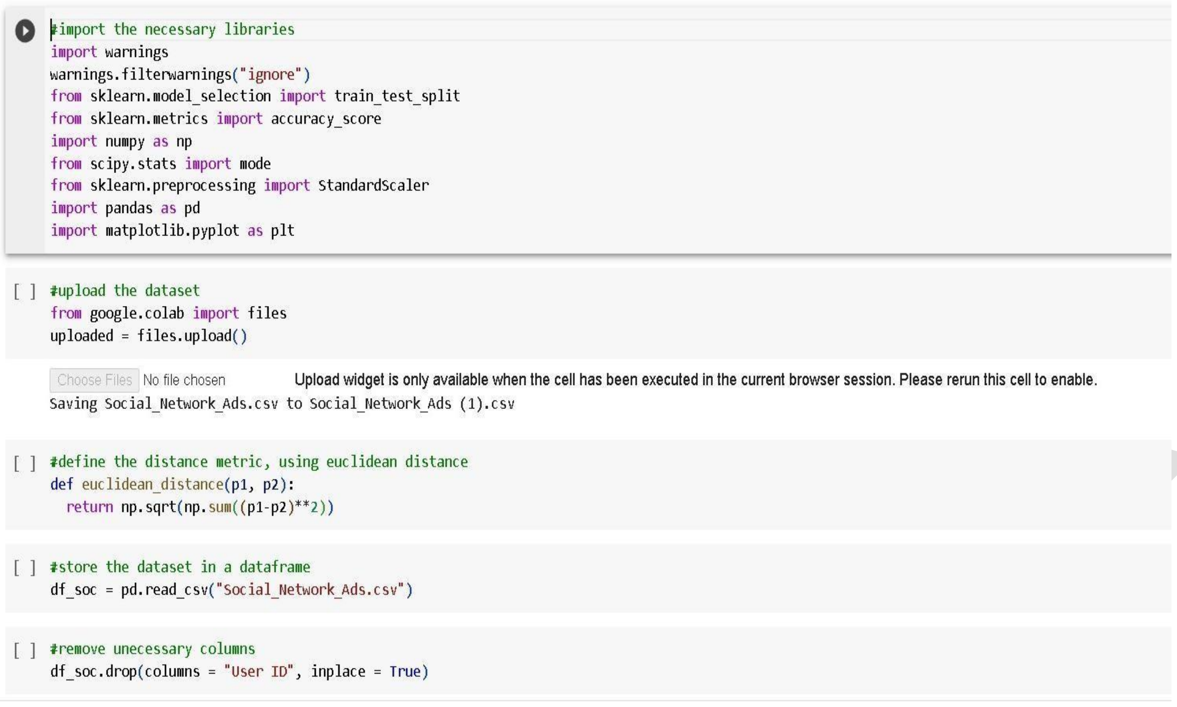
**THINGS TO NOTE**

The distance metric and the value of k are critical parameters that can have an enormous effect on the accuracy of the model.Because k-NN is sensitive to feature scale, feature scaling is frequently required.

The computational cost of the technique grows with the quantity of the training dataset since it must calculate distances to all data points.

It is critical to manage ties (where different classes have the same greatest frequency) and circumstances when the ideal k value is not evident.

## CODE



```
#import the necessary libraries
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
from scipy.stats import mode
from sklearn.preprocessing import StandardScaler
import pandas as pd
import matplotlib.pyplot as plt
```

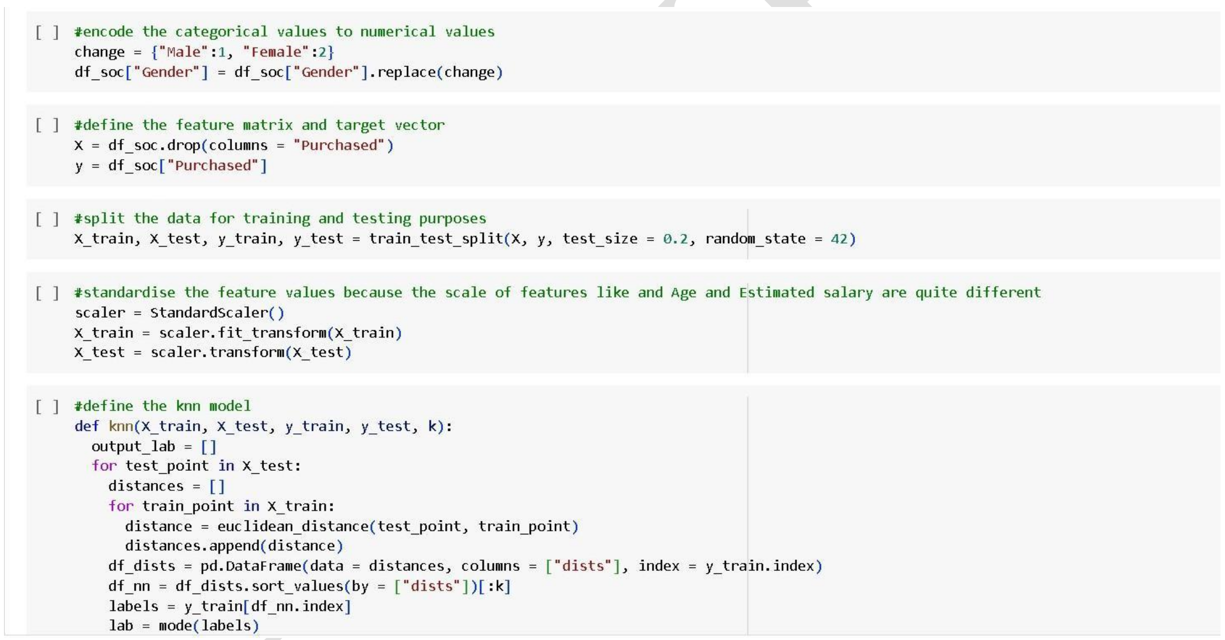
```
[ ] #upload the dataset
    from google.colab import files
    uploaded = files.upload()
```

```
Choose Files No file chosen        Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving Social_Network_Ads.csv to Social_Network_Ads (1).csv
```

```
[ ] #define the distance metric, using euclidean distance
    def euclidean_distance(p1, p2):
      return np.sqrt(np.sum((p1-p2)**2))
```

```
[ ] #store the dataset in a dataframe
    df_soc = pd.read_csv("Social_Network_Ads.csv")
```

```
[ ] #remove unecessary columns
    df_soc.drop(columns = "User ID", inplace = True)
```



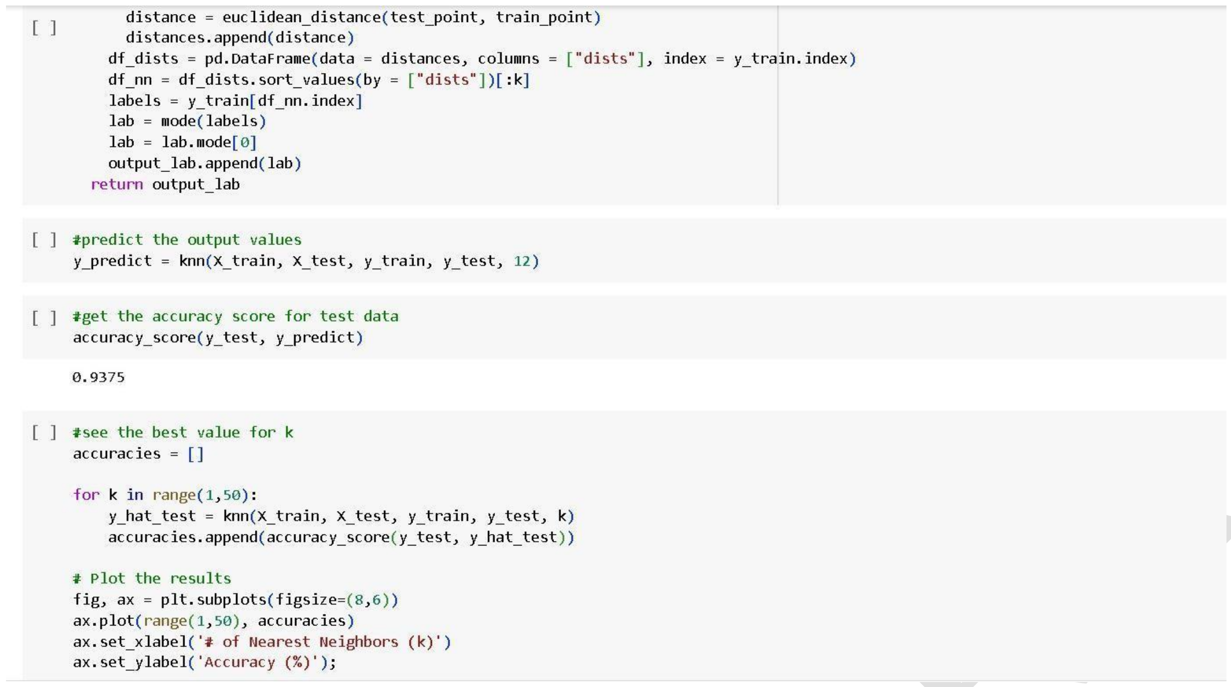
```
[ ] #encode the categorical values to numerical values
    change = {"Male":1, "Female":2}
    df_soc["Gender"] = df_soc["Gender"].replace(change)
```

```
[ ] #define the feature matrix and target vector
    X = df_soc.drop(columns = "Purchased")
    y = df_soc["Purchased"]
```

```
[ ] #split the data for training and testing purposes
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
[ ] #standardise the feature values because the scale of features like and Age and Estimated salary are quite different
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)
```

```
[ ] #define the knn model
    def knn(X_train, X_test, y_train, y_test, k):
      output_lab = []
      for test_point in X_test:
        distances = []
        for train_point in X_train:
          distance = euclidean_distance(test_point, train_point)
          distances.append(distance)
        df_dists = pd.DataFrame(data = distances, columns = ["dists"], index = y_train.index)
        df_nn = df_dists.sort_values(by = ["dists"])[:k]
        labels = y_train[df_nn.index]
        lab = mode(labels)
```

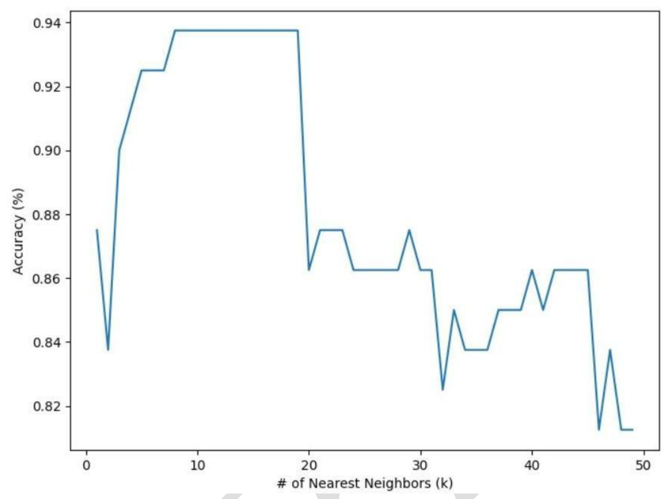


```
[ ]       distance = euclidean_distance(test_point, train_point)
          distances.append(distance)
        df_dists = pd.DataFrame(data = distances, columns = ["dists"], index = y_train.index)
        df_nn = df_dists.sort_values(by = ["dists"])[:k]
        labels = y_train[df_nn.index]
        lab = mode(labels)
        lab = lab.mode[0]
        output_lab.append(lab)
      return output_lab
```

```
[ ] #predict the output values
    y_predict = knn(X_train, X_test, y_train, y_test, 12)
```

```
[ ] #get the accuracy score for test data
    accuracy_score(y_test, y_predict)

    0.9375
```

```
[ ] #see the best value for k
    accuracies = []

    for k in range(1,50):
        y_hat_test = knn(X_train, X_test, y_train, y_test, k)
        accuracies.append(accuracy_score(y_test, y_hat_test))

    # Plot the results
    fig, ax = plt.subplots(figsize=(8,6))
    ax.plot(range(1,50), accuracies)
    ax.set_xlabel('# of Nearest Neighbors (k)')
    ax.set_ylabel('Accuracy (%)');
```



**SUMMARY**

The code creates a k-Nearest Neighbours (k-NN) classification model from the ground up. It has functions for computing Euclidean distances between data points as well as k-NN
classification. The code is deficient in dataset loading and splitting, which must be added individually. The knn function iterates through the test data, computes distances to the
training data, selects the k-nearest neighbours, and assigns the mode (most frequent) label as the

prediction. The accuracy comes out to be 0.9375.

**Random Forest**

The Random Forest algorithm is a collaborative learning approach that can be used for classification as well as regression applications. During training, it creates several decision trees and integrates their predictions to create a more robust and accurate model. Random Forest is well-known for its great predictive performance, resistance to overfitting, and adaptability to a broad variety of datasets.

**WORKING**

Data Preparation: Begin with a labelled dataset that includes features and goal parameters.

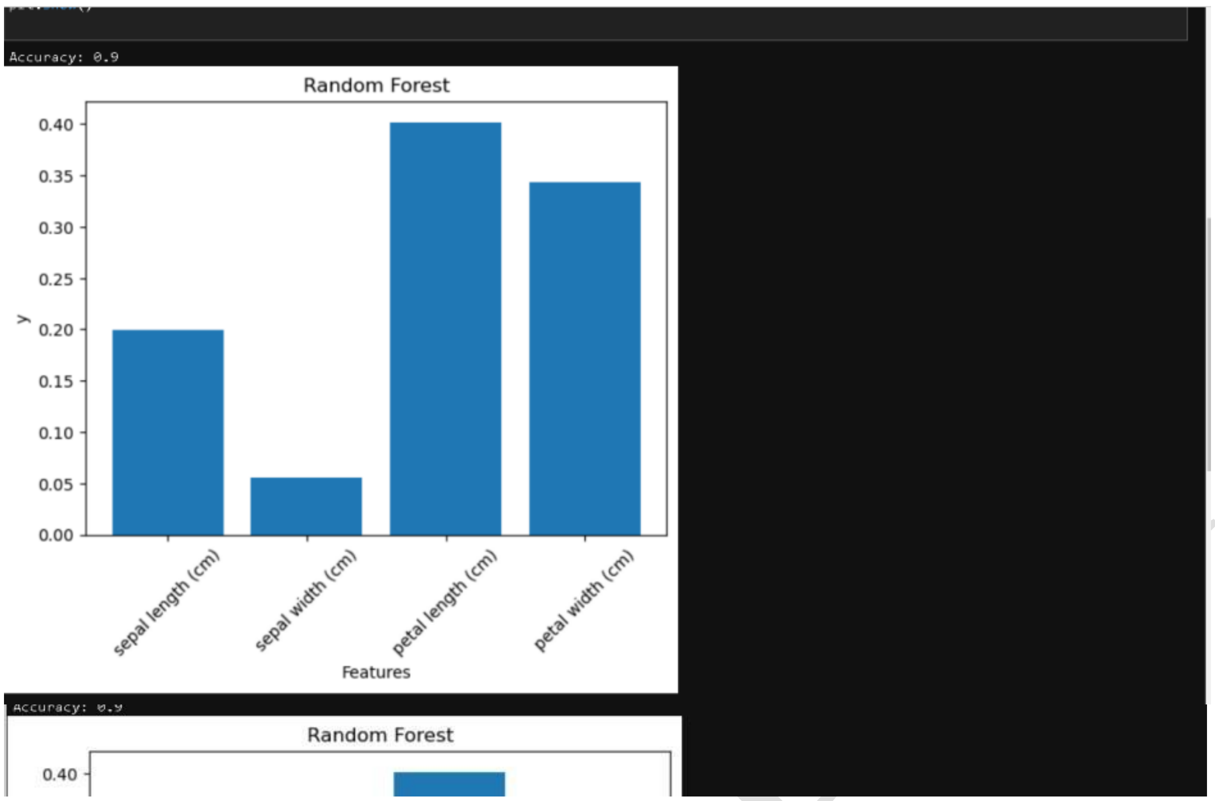
**Bootstrapping:** a randomly chosen sample is generated from the initial dataset for each tree in the Random Forest. This is referred to as bootstrapping, and it generates several subsets of data for each tree.

**Feature Randomization:** a randomly selected set of features is taken into account for each split when generating each decision tree. This randomization of features adds variation to the trees while decreasing the correlation between them.

**Building Decision Trees:** Using the selected characteristics, decision trees are built for each group of data. Repetitively, the trees are created by picking the best split at each node based on parameters such as Gini impurity for classification and mean squared error for regression.

A majority vote among the individual tree forecasts is used to make the final prediction in classification tasks. It is the average of the tree predictions for regression tasks.

The forecasts from all the trees in the Random Forest are merged to generate the final prediction. When contrasted to single decision trees, this combined paradigm frequently results in enhanced accuracy and generalization.

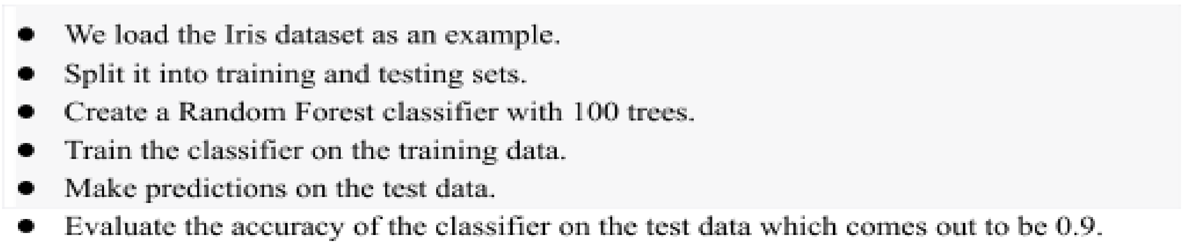


```
Accuracy: 0.9
```

```
Accuracy: 0.9
```

**CODE**

**SUMMARY**



- We load the Iris dataset as an example.
- Split it into training and testing sets.
- Create a Random Forest classifier with 100 trees.
- Train the classifier on the training data.
- Make predictions on the test data.
- Evaluate the accuracy of the classifier on the test data which comes out to be 0.9.

**Naïve Bayes**

The Naive Bayes algorithm is a probabilistic classification technique that is based on the theorem of Bayes. It is referred described as "naive" because it makes a strong assumption of feature independence, which indicates that the existence a specific attribute is

independent to the availability another. Despite this simplistic assumption, Naive Bayes can be surprisingly effective in a variety of classification problems, particularly text

categorization and spam detection.

**WOKING**

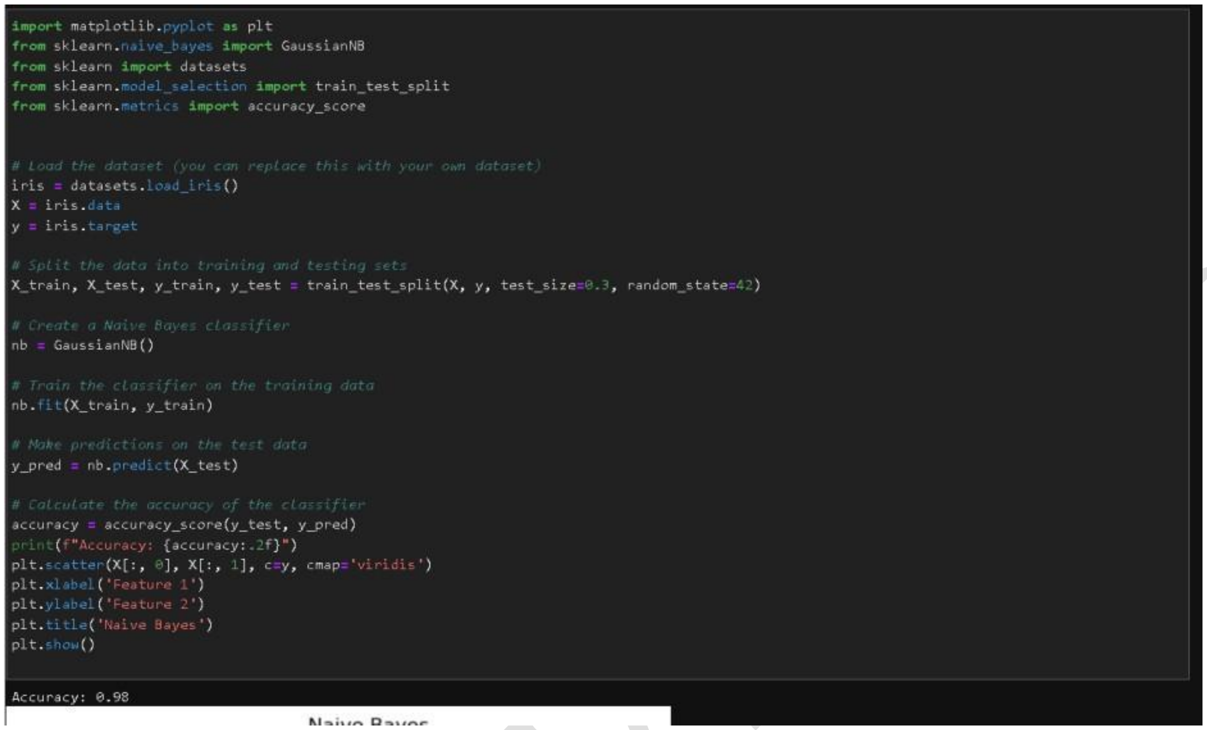
**_Data Preparation**: Begin with a labelled dataset that includes features and goal variables.

Naive Bayes determines the estimated likelihood of every class (target variable) and the conditional probabilities of every characteristic given each class throughout the training phase. This entails determining the number of feature-class combinations that occur in the training data.

**Classification:**

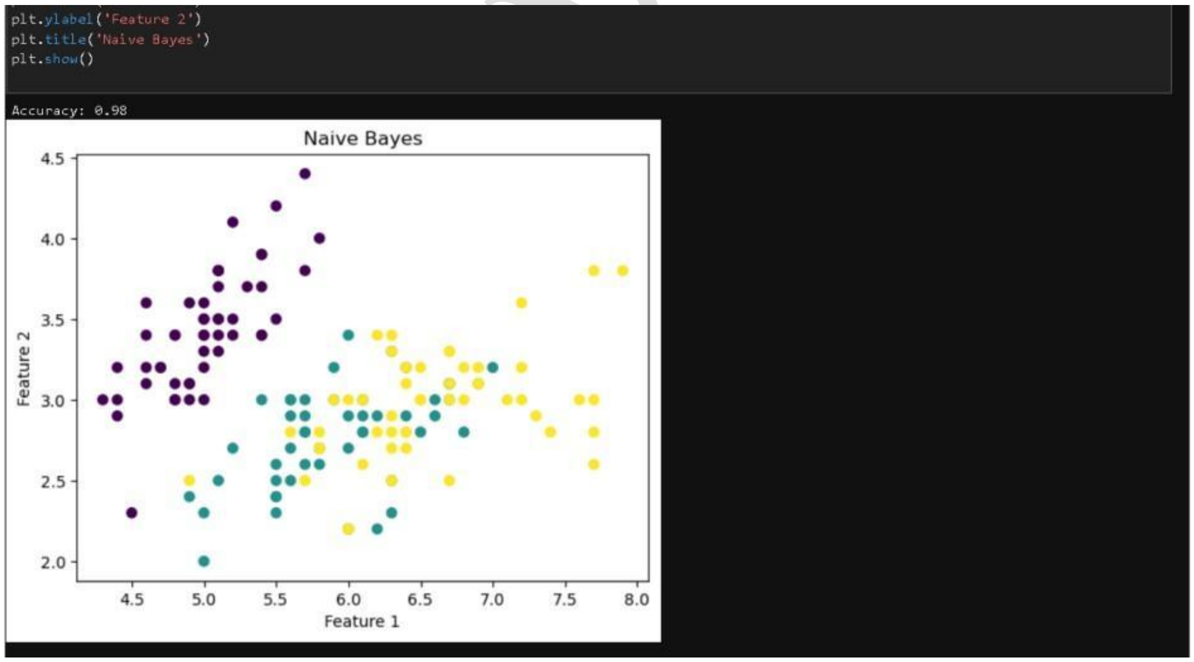
Applying Bayes' theorem, Naive Bayes determines the posterior probability of each class given the characteristics seen for a particular data point and collection of features.
It predicts the class with the highest posterior probability as the anticipated class for the data point.

**CODE**



```
import matplotlib.pyplot as plt
from sklearn.naive_bayes import GaussianNB
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score


# Load the dataset (you can replace this with your own dataset)
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Create a Naive Bayes classifier
nb = GaussianNB()

# Train the classifier on the training data
nb.fit(X_train, y_train)

# Make predictions on the test data
y_pred = nb.predict(X_test)

# Calculate the accuracy of the classifier
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
plt.scatter(X[:, 0], X[:, 1], c=y, cmap='viridis')
plt.xlabel('Feature 1')
plt.ylabel('Feature 2')
plt.title('Naive Bayes')
plt.show()
```

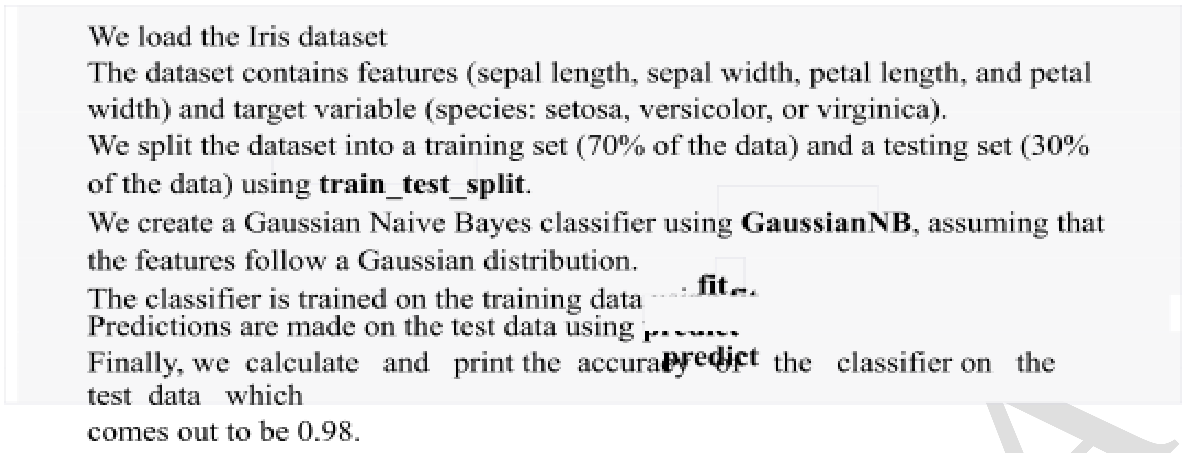
```
Accuracy: 0.98
```



```
plt.ylabel('Feature 2')
plt.title('Naive Bayes')
plt.show()
```

```
Accuracy: 0.98
```

## SUMMARY


We load the Iris dataset
The dataset contains features (sepal length, sepal width, petal length, and petal width) and target variable (species: setosa, versicolor, or virginica).
We split the dataset into a training set (70% of the data) and a testing set (30% of the data) using **train_test_split**.
We create a Gaussian Naive Bayes classifier using **GaussianNB**, assuming that the features follow a Gaussian distribution.
The classifier is trained on the training data **fit**.
Predictions are made on the test data using **predict**
Finally, we calculate and print the accuracy of the classifier on the test data which
comes out to be 0.98.

## CONCLUSION

In conclusion, the intersection of artificial intelligence (AI) and machine learning (ML) with power quality analytics holds immense promise for the future of our energy infrastructure. As our dependence on electricity continues to grow, ensuring the stable and reliable supply of high-quality electrical power becomes paramount. The development and application of AI/ML techniques in this domain have the potential to revolutionize the way we assess, monitor, and maintain power quality.

This research report has articulated several pivotal objectives and aspirations in harnessing AI/ML for power quality analysis:

**Elevated Precision:** The fusion of AI/ML algorithms holds the promise of elevating the precision of power quality assessment, enabling the identification and characterization of electrical irregularities and anomalies with unparalleled accuracy.

**Real-time Vigilance:** The prospect of real-time power quality monitoring, facilitated by AI/ML technologies, stands as a linchpin in our ability to enact timely interventions and maintenance, thus forestalling potential complications before they escalate.

**Insights from Data:** Through the analysis of extensive datasets, our aim is to uncover concealed patterns and interconnections within power quality data, endowing operators, utilities, and power grid researchers with invaluable insights to guide their decision-making processes.

As we journey forward, we remain fervently optimistic about the transformative capabilities of AI/ML in enhancing power quality, thus sculpting a more robust and sustainable energy horizon for all to embrace.