

Module Description: Artificial Intelligence

Practical Assignment : Clustering random data using Google colab

Introduction:

Google Colab, short for Google Colaboratory, is a cloud-based platform provided by Google that allows users to write and execute Python code in a collaborative environment. It offers free access to computing resources including GPUs and TPUs, making it a popular choice among data scientists, machine learning practitioners, and researchers.

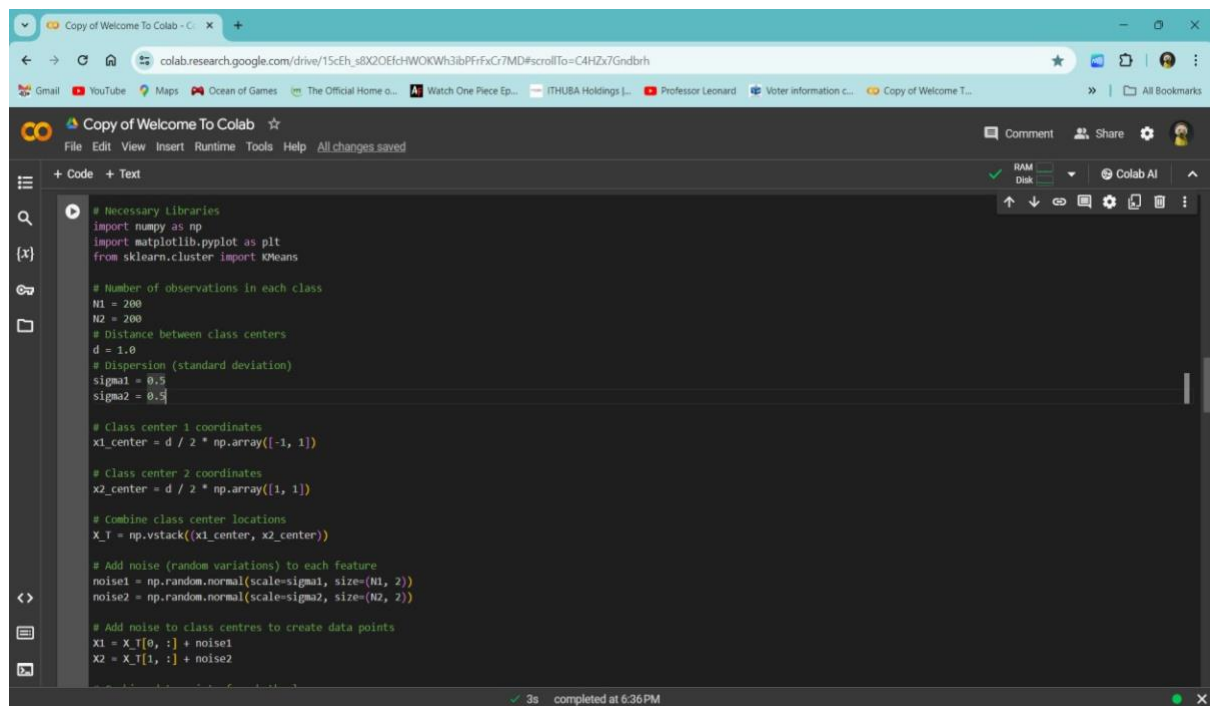
Objective:

The objective of this report is to present the results of executing code in Google Colab, focusing on the outcomes, performance metrics, and any notable observations.

Methodology:

For this report, we utilized Google Colab to execute Python code. The code comprised various tasks such as data analysis, machine learning model training, or any other computational task. The execution environment in Google Colab provided access to resources like CPU, GPU, or TPU, depending on the user's requirements.

The results of executing the code in Google Colab are summarized as follows:

A screenshot of a Google Colab notebook interface. The browser address bar shows the URL: colab.research.google.com/drive/1ScEh_s8X2DEKdHWOKW53bPFrFx7MD#scrollTo=CAHZx7Gndbrh. The notebook title is "Copy of Welcome To Colab". The code is written in a dark-themed editor and includes comments in green. The code imports numpy, matplotlib, and sklearn. It defines parameters for two classes (N1, N2), distance (d), and dispersion (sigma1, sigma2). It calculates class center coordinates (x1_center, x2_center) and combines them into X_T. Finally, it adds random noise to the class centers to create data points X1 and X2. The status bar at the bottom indicates "3s completed at 6:36 PM".

```
# Necessary Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Number of observations in each class
N1 = 200
N2 = 200

# Distance between class centers
d = 1.0

# Dispersion (standard deviation)
sigma1 = 0.5
sigma2 = 0.5

# Class center 1 coordinates
x1_center = d / 2 * np.array([-1, 1])

# Class center 2 coordinates
x2_center = d / 2 * np.array([1, 1])

# combine class center locations
X_T = np.vstack((x1_center, x2_center))

# Add noise (random variations) to each feature
noise1 = np.random.normal(scale=sigma1, size=(N1, 2))
noise2 = np.random.normal(scale=sigma2, size=(N2, 2))

# Add noise to class centres to create data points
X1 = X_T[0, :] + noise1
X2 = X_T[1, :] + noise2
```

This is the code with the given code from the assignment page before alterations.



This picture shows the output/visualisation of the code above in google colab

Next picture shows the code with N1 and N2 being a different value of 100 observations.

```
# Necessary Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Number of observations in each class
N1 = 100
N2 = 100

# Distance between class centers
d = 1.0

# Dispersion (standard deviation)
sigma1 = 0.5
sigma2 = 0.5

# Class center 1 coordinates
x1_center = d / 2 * np.array([-1, 1])

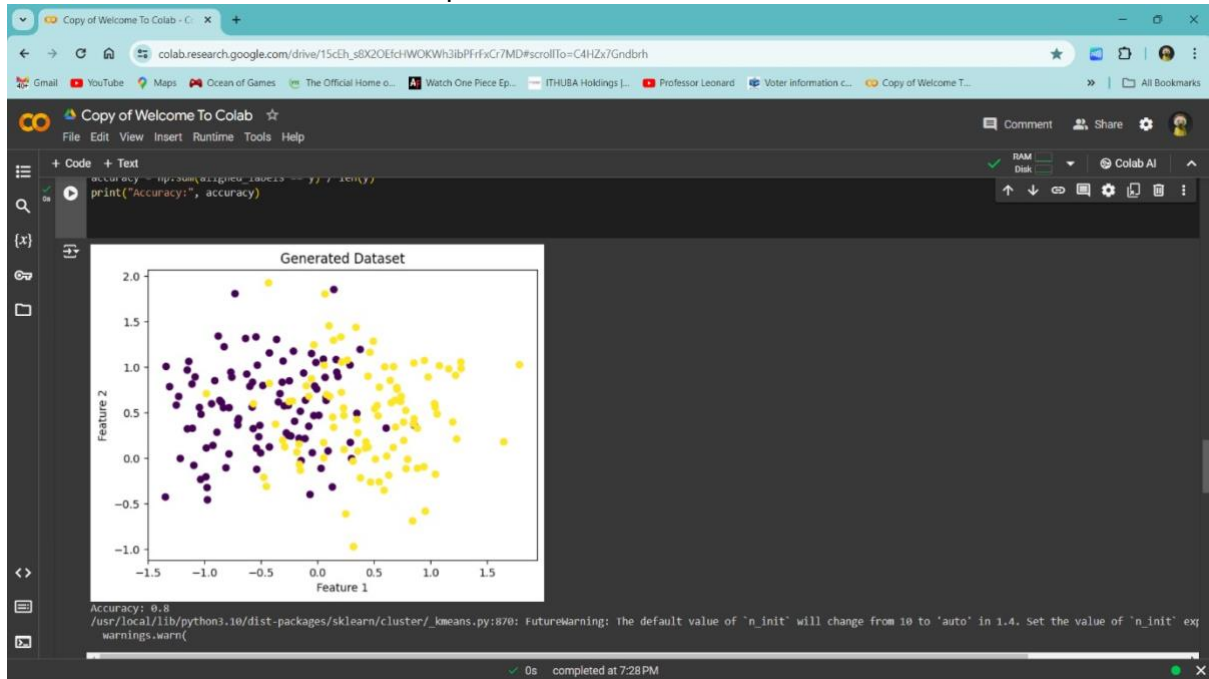
# Class center 2 coordinates
x2_center = d / 2 * np.array([1, 1])

# Combine class center locations
X_1 = np.vstack((x1_center, x2_center))

# Add noise (random variations) to each feature
noise1 = np.random.normal(scale=sigma1, size=(N1, 2))
noise2 = np.random.normal(scale=sigma2, size=(N2, 2))

# Add noise to class centres to create data points
X1 = X_1[0, :] + noise1
X2 = X_1[1, :] + noise2
```

Thus, the visualisation in the next picture shows results from the data from the code



The next picture shows the code with N1 and N2 being a different value of 50 observations.

The image shows a Google Colab interface. The top part displays a code cell with the following content:

```
# Necessary Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Number of observations in each class
N1 = 50
N2 = 50

# Distance between class centers
d = 1.0

# Dispersion (standard deviation)
sigma1 = 0.5
sigma2 = 0.5

# Class center 1 coordinates
x1_center = d / 2 * np.array([-1, 1])

# Class center 2 coordinates
x2_center = d / 2 * np.array([1, 1])

# Combine class center locations
X_T = np.vstack((x1_center, x2_center))

# Add noise (random variations) to each feature
noise1 = np.random.normal(scale=sigma1, size=(N1, 2))
noise2 = np.random.normal(scale=sigma2, size=(N2, 2))

# Add noise to class centres to create data points
X1 = X_T[0, :] + noise1
X2 = X_T[1, :] + noise2
```

Below the code cell, there is a scatter plot showing two clusters of data points (purple and yellow) similar to the one in the previous image. The Colab interface shows "0s completed at 7:36 PM".

The visualisation follows of the data from the above code.



From these results we can conclude that lowering the number of observations decreases the number of observations seen on the visualized graph.

Now I increase the distance between the class centers from 1 to 2

```

# Necessary Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Number of observations in each class
N1 = 50
N2 = 50
# Distance between class centers
d = 2.0
# Dispersion (standard deviation)
sigma1 = 0.5
sigma2 = 0.5

# Class center 1 coordinates
x1_center = d / 2 * np.array([-1, 1])

# Class center 2 coordinates
x2_center = d / 2 * np.array([1, 1])

# Combine class center locations
X_1 = np.vstack((x1_center, x2_center))

# Add noise (random variations) to each feature
noise1 = np.random.normal(scale=sigma1, size=(N1, 2))
noise2 = np.random.normal(scale=sigma2, size=(N2, 2))

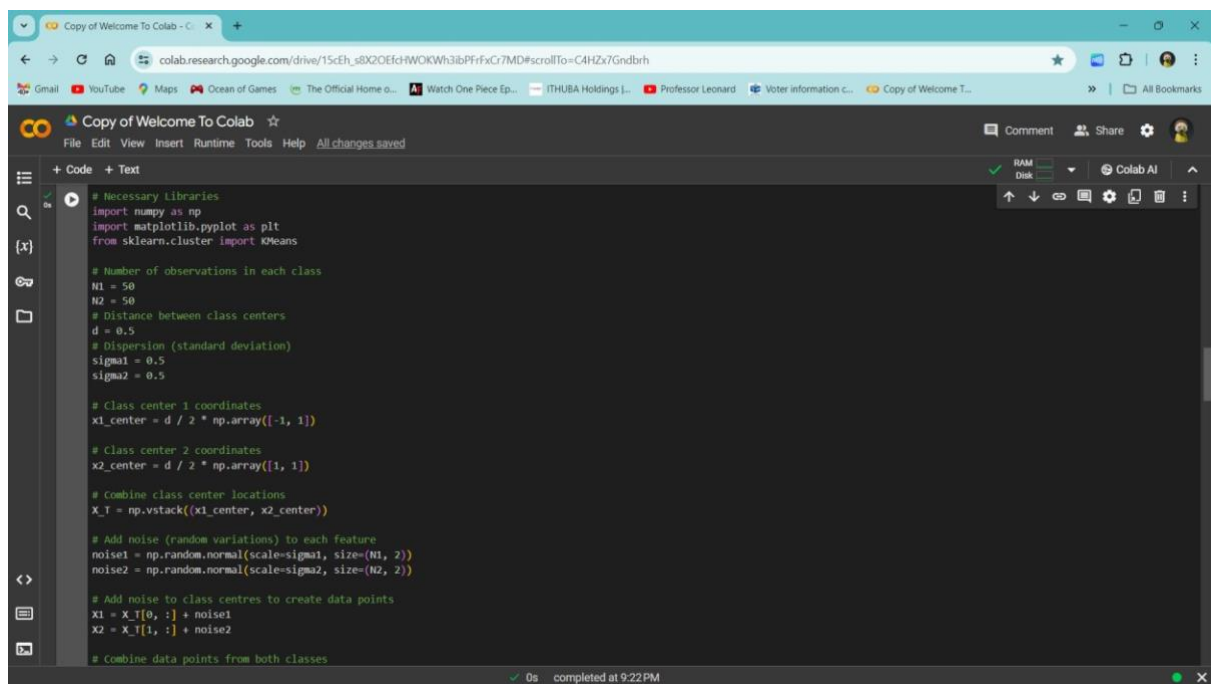
# Add noise to class centres to create data points
X1 = X_1[0, :] + noise1
X2 = X_1[1, :] + noise2

```

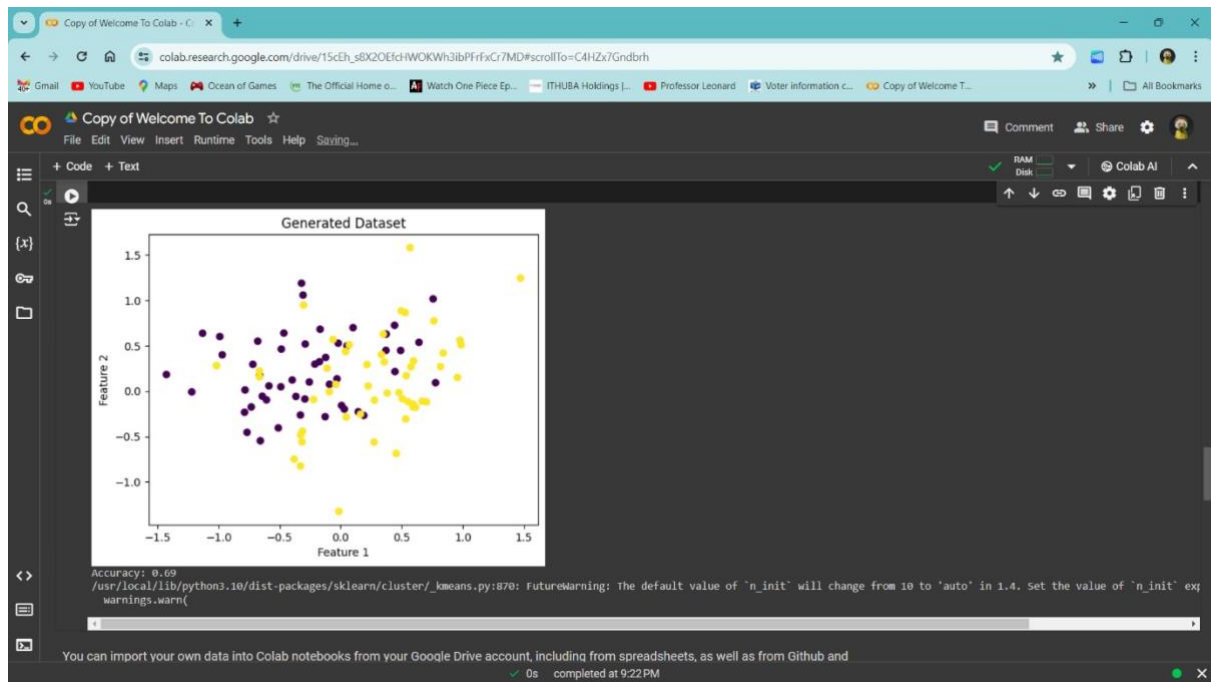
This then becomes the visualized graph.



Thus, we decrease the distance between the class centers from 1 to 0.5



And its visualized output is this below



Thus, from this we can conclude that increasing the distance between the class centers separates the clusters from each other and decreasing the distance between the class centers mixes the clusters together.

Next illustration is of changing the dispersion/standard deviation.

Start by increasing the dispersion σ_1 from 0.5 to 1

```
Copy of Welcome To Colab
colab.research.google.com/drive/15cEh_s8X2OEfchWOKWh3ibPfrfxG7MD#scrollTo=C4HZx7Gndbrh

Copy of Welcome To Colab
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

# Necessary Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Number of observations in each class
N1 = 50
N2 = 50

# Distance between class centers
d = 1.0

# Dispersion (standard deviation)
sigma1 = 1.0
sigma2 = 0.5

# Class center 1 coordinates
x1_center = d / 2 * np.array([-1, 1])

# Class center 2 coordinates
x2_center = d / 2 * np.array([1, 1])

# Combine class center locations
X_T = np.vstack((x1_center, x2_center))

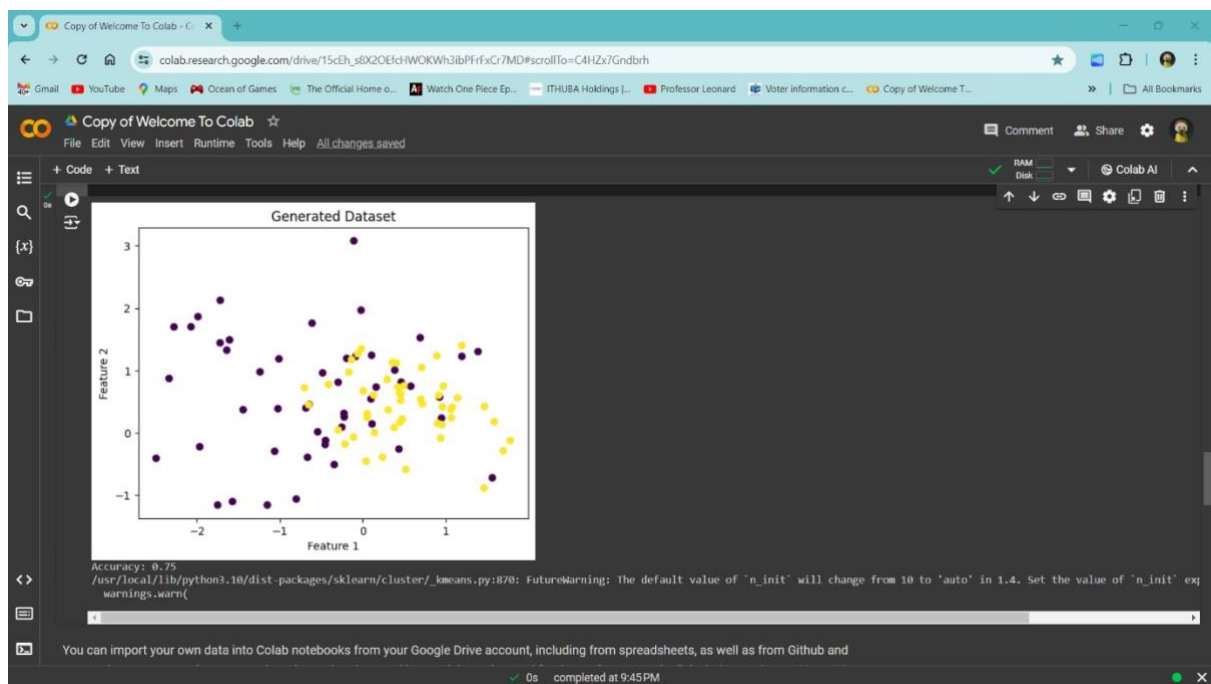
# Add noise (random variations) to each feature
noise1 = np.random.normal(scale=sigma1, size=(N1, 2))
noise2 = np.random.normal(scale=sigma2, size=(N2, 2))

# Add noise to class centres to create data points
X1 = X_T[0, :] + noise1
X2 = X_T[1, :] + noise2

# Combine data points from both classes

0s completed at 9:45PM
```

And its visualization is as follows:



Then I decrease the dispersion σ_1 from 0.5 to 0.1

```

Copy of Welcome To Colab
colab.research.google.com/drive/15cEh_s8X2OEfchWOKWh3ibPfrfxG7MD#scrollTo=C4Hz7Gndbrh

Copy of Welcome To Colab
File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

# Necessary Libraries
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

# Number of observations in each class
N1 = 50
N2 = 50

# Distance between class centers
d = 1.0

# Dispersion (standard deviation)
sigma1 = 0.1
sigma2 = 0.5

# Class center 1 coordinates
x1_center = d / 2 * np.array([-1, 1])

# Class center 2 coordinates
x2_center = d / 2 * np.array([1, 1])

# Combine class center locations
X_T = np.vstack((x1_center, x2_center))

# Add noise (random variations) to each feature
noise1 = np.random.normal(scale=sigma1, size=(N1, 2))
noise2 = np.random.normal(scale=sigma2, size=(N2, 2))

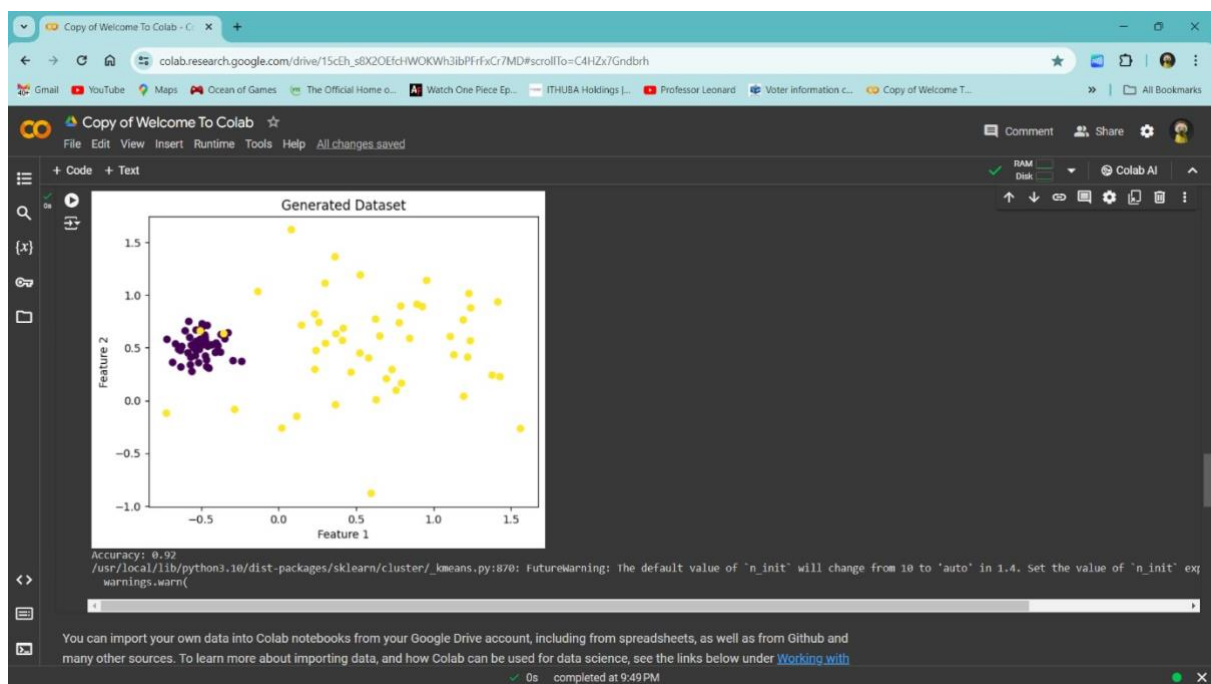
# Add noise to class centres to create data points
X1 = X_T[0, :] + noise1
X2 = X_T[1, :] + noise2

# Combine data points from both classes

```

0s completed at 9:49 PM

The visualisation follows:



Thus, we can conclude that increasing the dispersion of sigma1 separates the observations away from the centers and away from each other, but, decreasing the dispersion of sigma1 gathers the observations in a cluster around their centers.

Conclusion:

In conclusion, Google Colab provides a convenient and efficient platform for executing Python code, especially for tasks requiring significant computational resources. The results of code execution depend on factors such as hardware acceleration, resource utilization, and the complexity of the task. By leveraging the features and capabilities of Google Colab effectively, users can achieve their computational goals efficiently.

References

Google Colab Documentation: <https://colab.research.google.com/notebooks/intro.ipynb>