**CSA4724-DEEP LEARNING FOR NUTRITION ANALYSIS**

**Prasanna.C(192124160)**

**22.06.24 (DAY 2)**

**4.**

**Program:**

import numpy as np

import matplotlib.pyplot as plt

def estimate\_coef(x, y):

# number of observations/points

n = np.size(x)

# mean of x and y vector

m\_x = np.mean(x)

m\_y = np.mean(y)

# calculating cross-deviation and deviation about x

SS\_xy = np.sum(y \* x) - n \* m\_y \* m\_x

SS\_xx = np.sum(x \* x) - n \* m\_x \* m\_x

# calculating regression coefficients

b\_1 = SS\_xy / SS\_xx

b\_0 = m\_y - b\_1 \* m\_x

return b\_0, b\_1

def plot\_regression\_line(x, y, b):

# plotting the actual points as scatter plot

plt.scatter(x, y, color="r", marker="o", s=30)

# predicted response vector

y\_pred = b[0] + b[1] \* x

# plotting the regression line

plt.plot(x, y\_pred, color="b")

# putting labels

plt.xlabel('x')

plt.ylabel('y')

# function to show plot

plt.show()

def main():

# observations / data

x = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])

y = np.array([1, 3, 2, 5, 7, 8, 8, 9, 10, 12])

# estimating coefficients

b = estimate\_coef(x, y)

print(f"Estimated coefficients:\nb\_0 = {b[0]} \nb\_1 = {b[1]}")

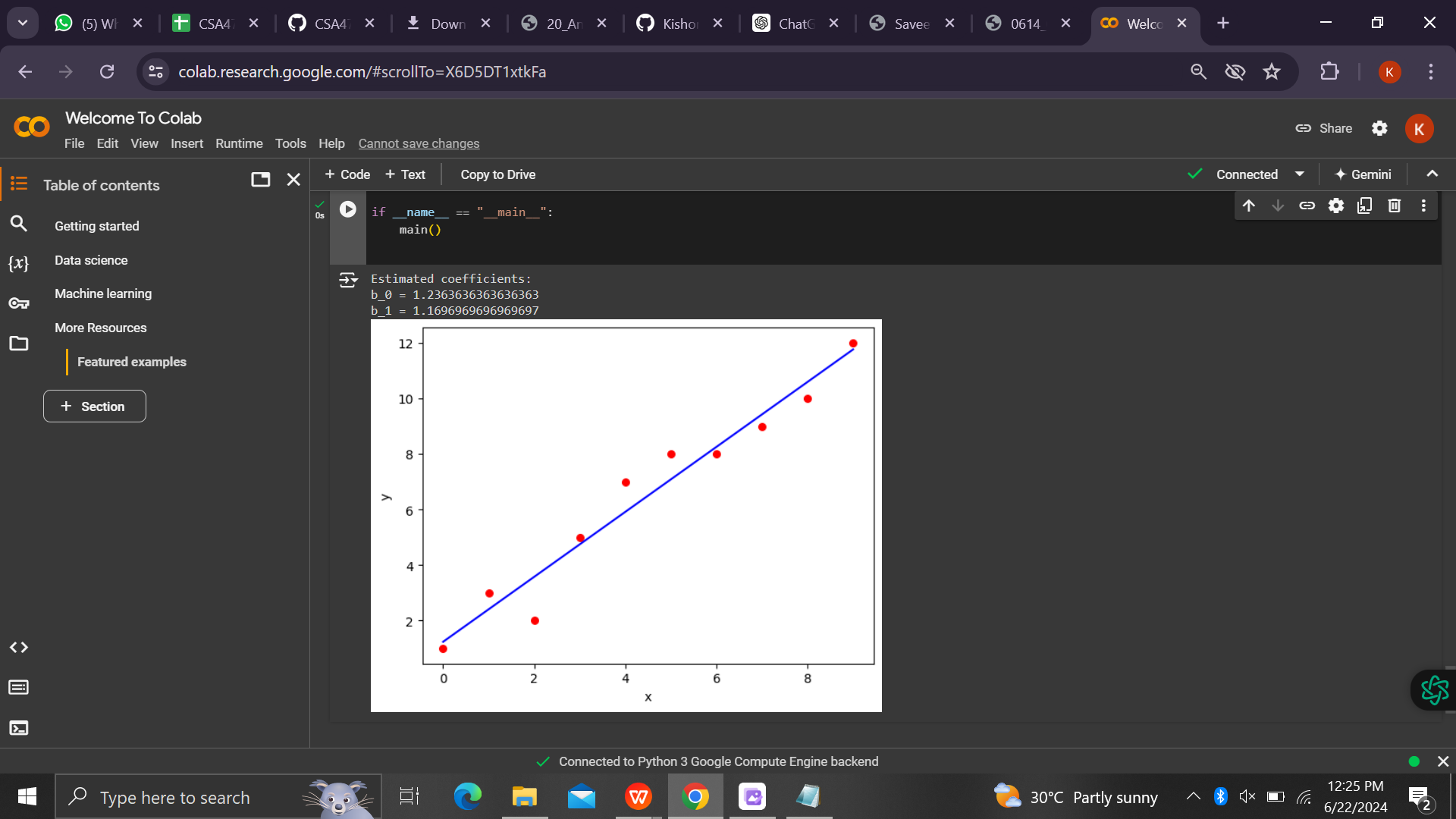
# plotting regression line

plot\_regression\_line(x, y, b)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output:**

****

**5**

**Program:**

import numpy as np

import matplotlib.pyplot as plt

def sigmoid(z):

return 1 / (1 + np.exp(-z))

# Define the range for z

z = np.arange(-1, 4, 0.3)

# Calculate the sigmoid values

sigmoid\_values = sigmoid(z)

# Plot the sigmoid function

plt.plot(z, sigmoid\_values)

plt.title('Visualization of the Sigmoid Function')

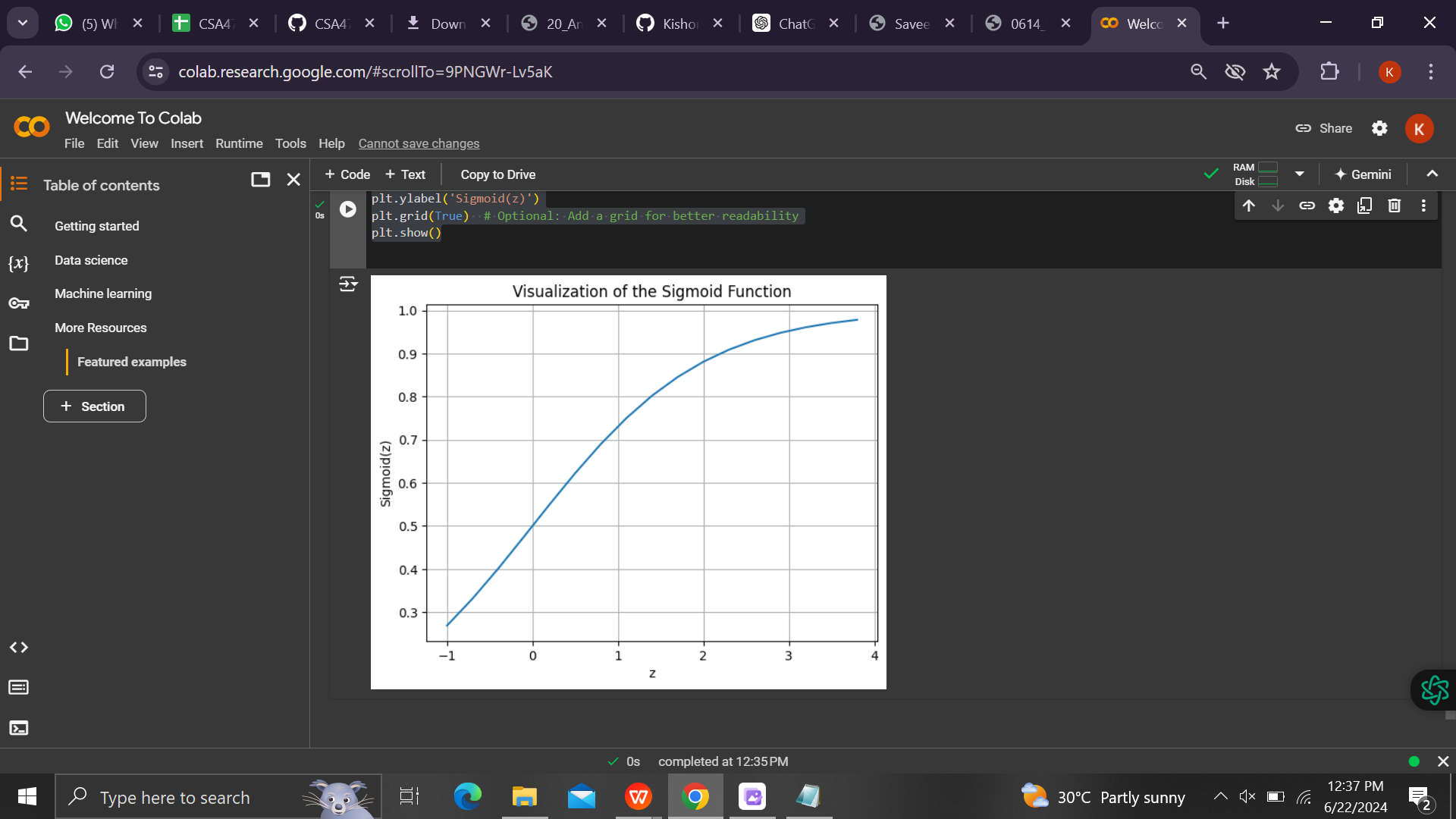
plt.xlabel('z')

plt.ylabel('Sigmoid(z)')

plt.grid(True) # Optional: Add a grid for better readability

plt.show()

**Output:**

****

**6(B)**

**Program:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix, accuracy\_score

from sklearn.datasets import load\_iris

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

# Checking the dataset structure

print("Dataset shape:", X.shape)

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=0)

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Training the Naive Bayes Classification model on the Training set

classifier = GaussianNB()

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Display the results (confusion matrix and accuracy)

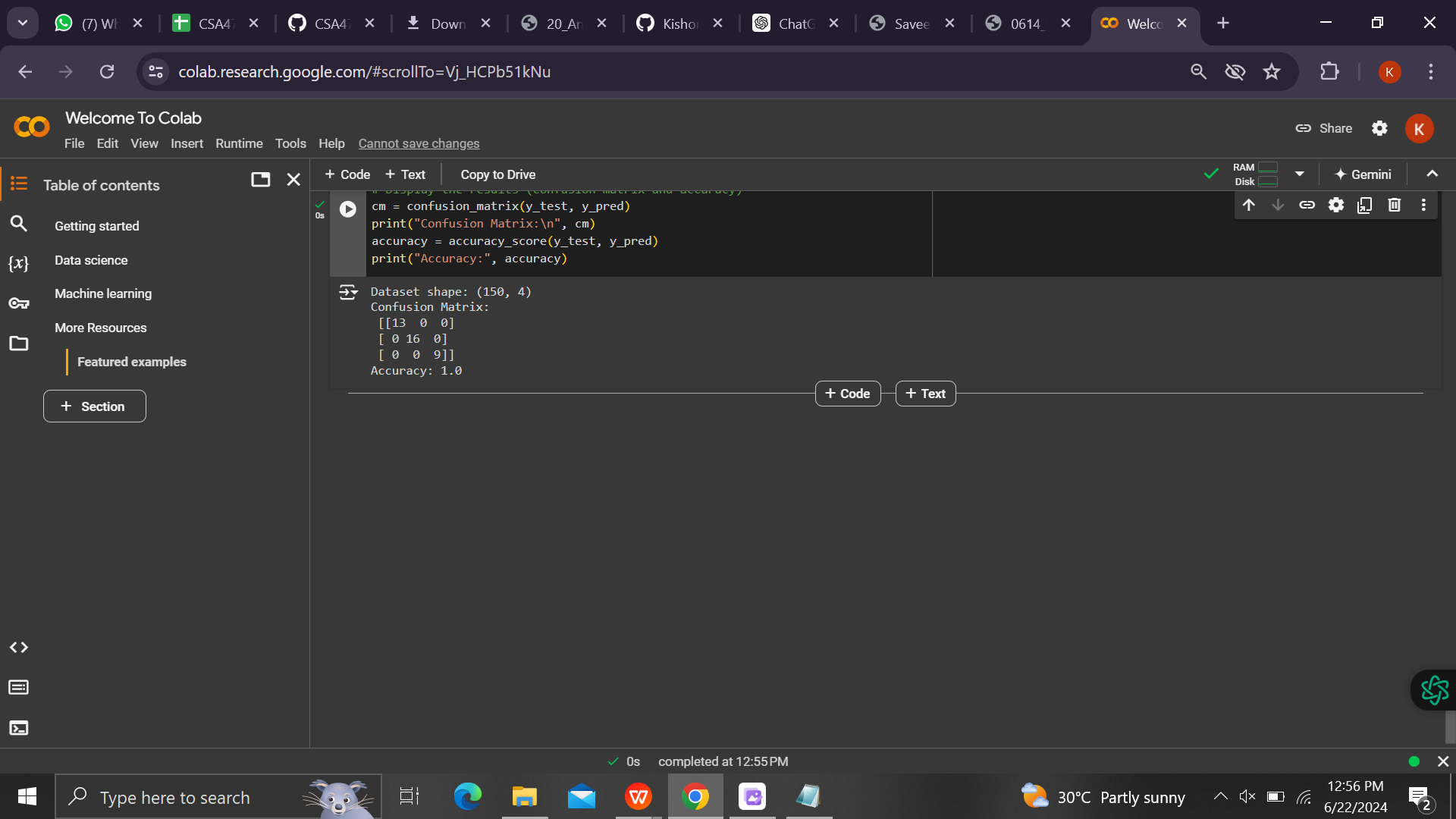
cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

**Output:**

****

**6(A)**

**Program:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score

from sklearn.datasets import load\_iris

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

feature\_names = iris.feature\_names

# Checking the dataset structure

print("Dataset shape:", X.shape)

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=42)

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Training the K-Nearest Neighbors (K-NN) Classification model on the Training set

classifier = KNeighborsClassifier(n\_neighbors=5, metric='minkowski', p=2)

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Display the results (confusion matrix and accuracy)

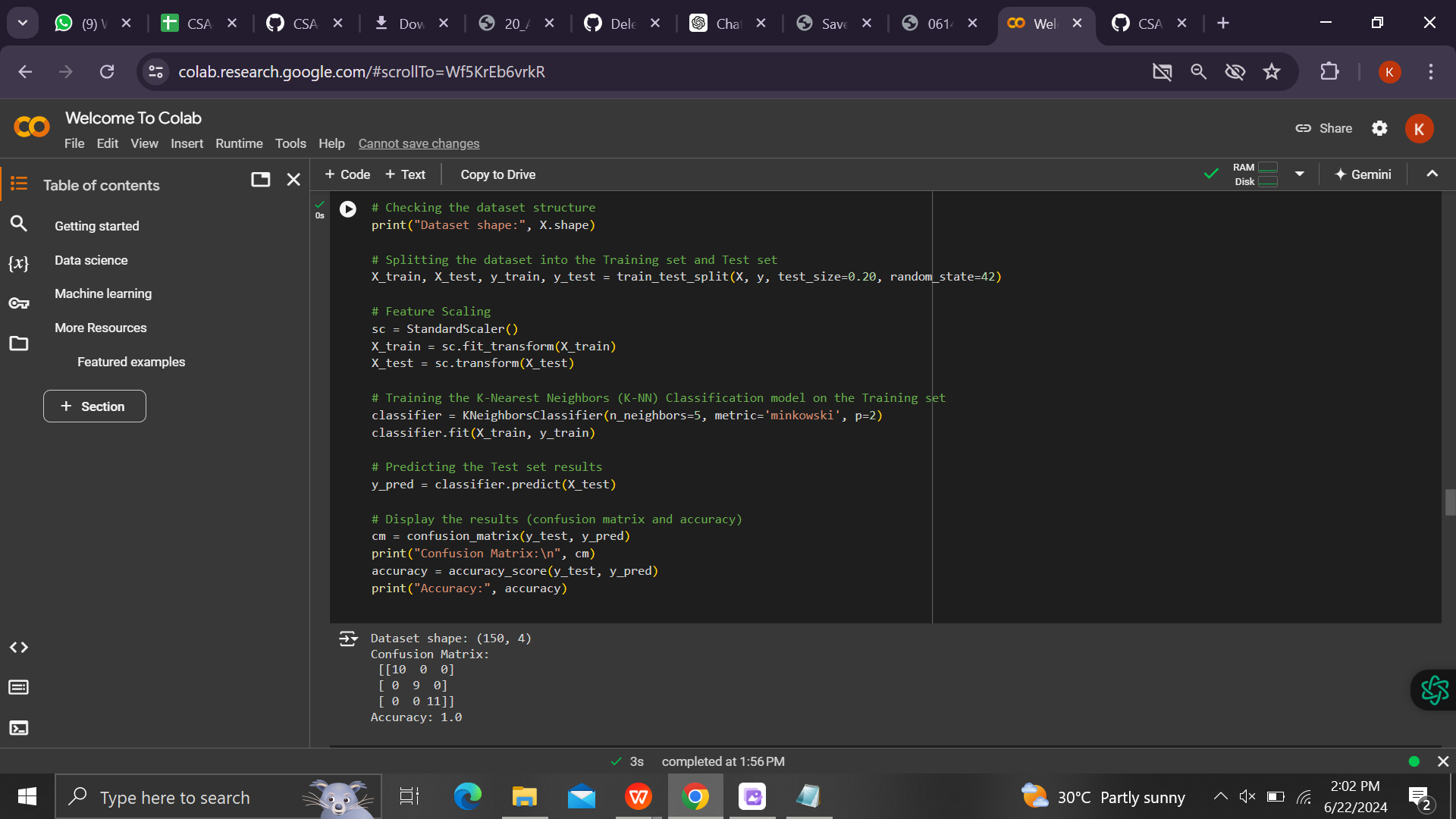
cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

**Output:**

****

**6(C)**

**Program:**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# Training the Logistic Regression model on the Training set

classifier = LogisticRegression(random\_state=0)

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Evaluating the performance of the model using confusion matrix and accuracy

cm = confusion\_matrix(y\_test, y\_pred)

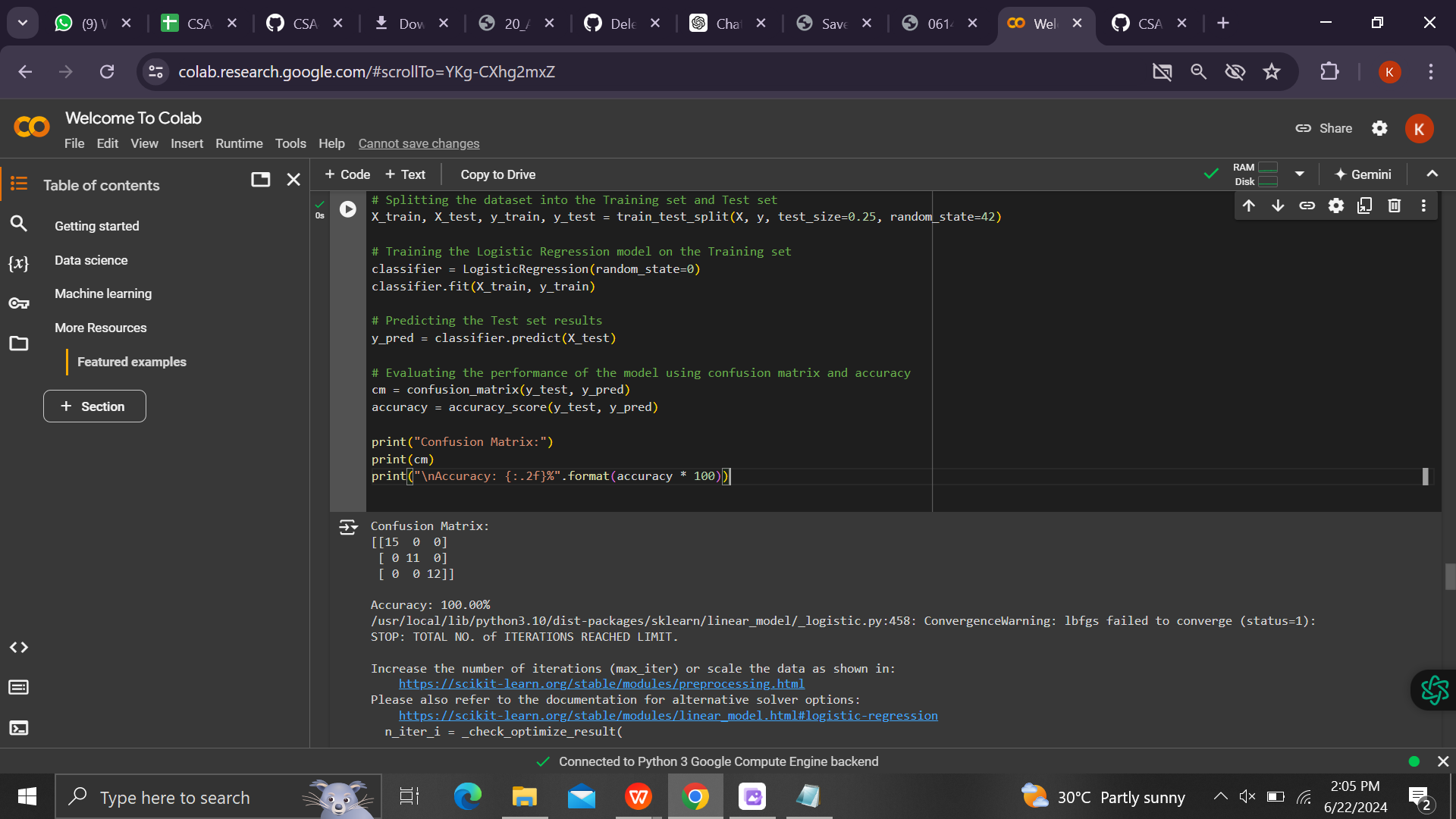
accuracy = accuracy\_score(y\_test, y\_pred)

print("Confusion Matrix:")

print(cm)

print("\nAccuracy: {:.2f}%".format(accuracy \* 100))

**Output:**

****

**6(D)**

**Program:**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.metrics import confusion\_matrix, accuracy\_score

import matplotlib.pyplot as plt

from sklearn.datasets import make\_classification

# Create a synthetic dataset

X, y = make\_classification(n\_samples=150, n\_features=4, n\_informative=3, n\_redundant=0, random\_state=8, shuffle=True)

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.20, random\_state=7)

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Training the Decision Tree Classification model on the Training set

classifier = DecisionTreeClassifier(criterion='entropy', random\_state=5)

classifier.fit(X\_train, y\_train)

# Display the Decision Tree

plt.figure(figsize=(10,10))

plot\_tree(classifier, filled=True, rounded=True)

plt.show()

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

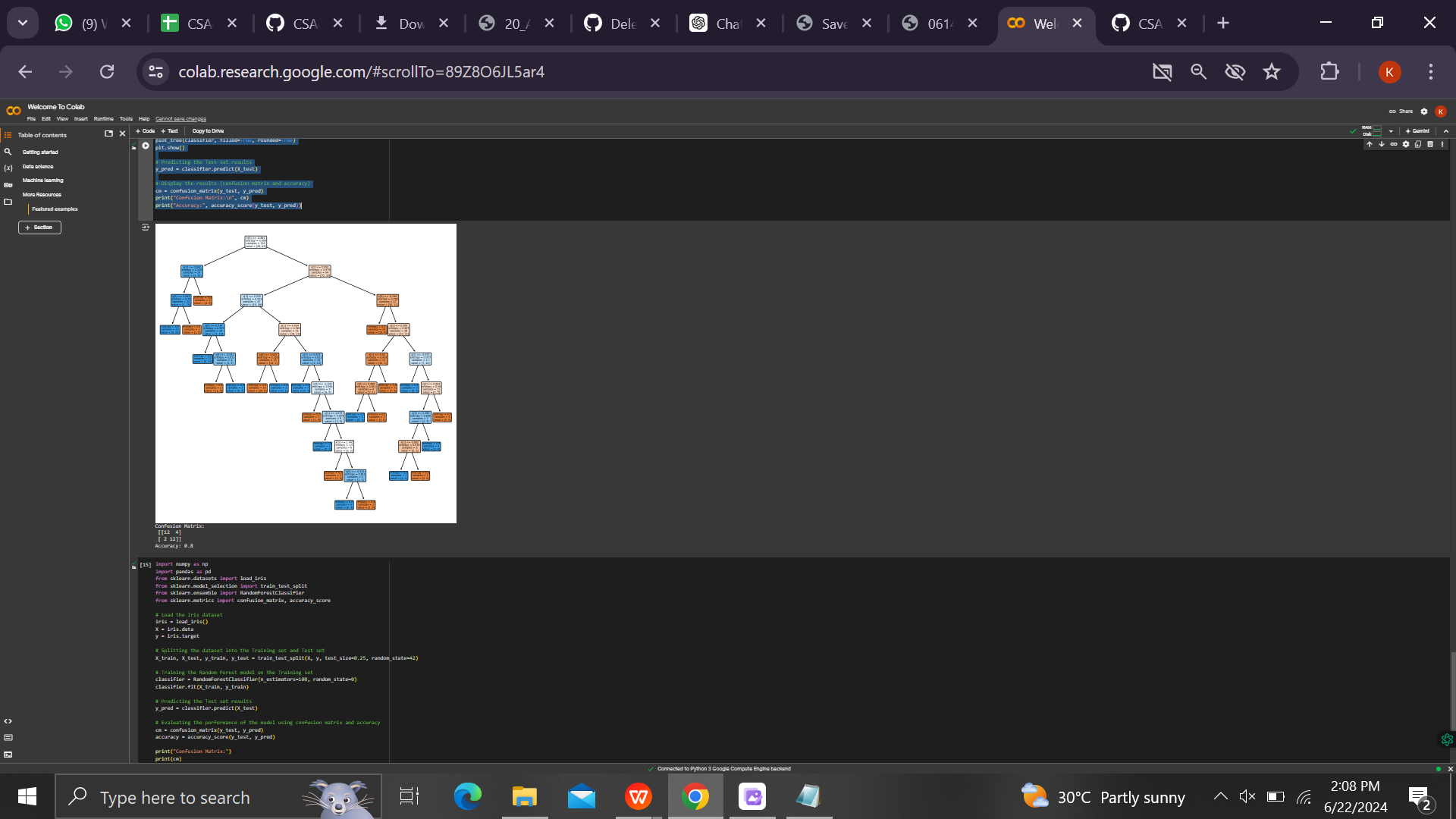
# Display the results (confusion matrix and accuracy)

cm = confusion\_matrix(y\_test, y\_pred)

print("Confusion Matrix:\n", cm)

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

**Output:**

****

**6(E)**

**Program:**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.metrics import confusion\_matrix, accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# Feature Scaling

sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train)

X\_test = sc.transform(X\_test)

# Training the SVM model on the Training set

classifier = SVC(kernel='linear', random\_state=0)

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Evaluating the performance of the model using confusion matrix and accuracy

cm = confusion\_matrix(y\_test, y\_pred)

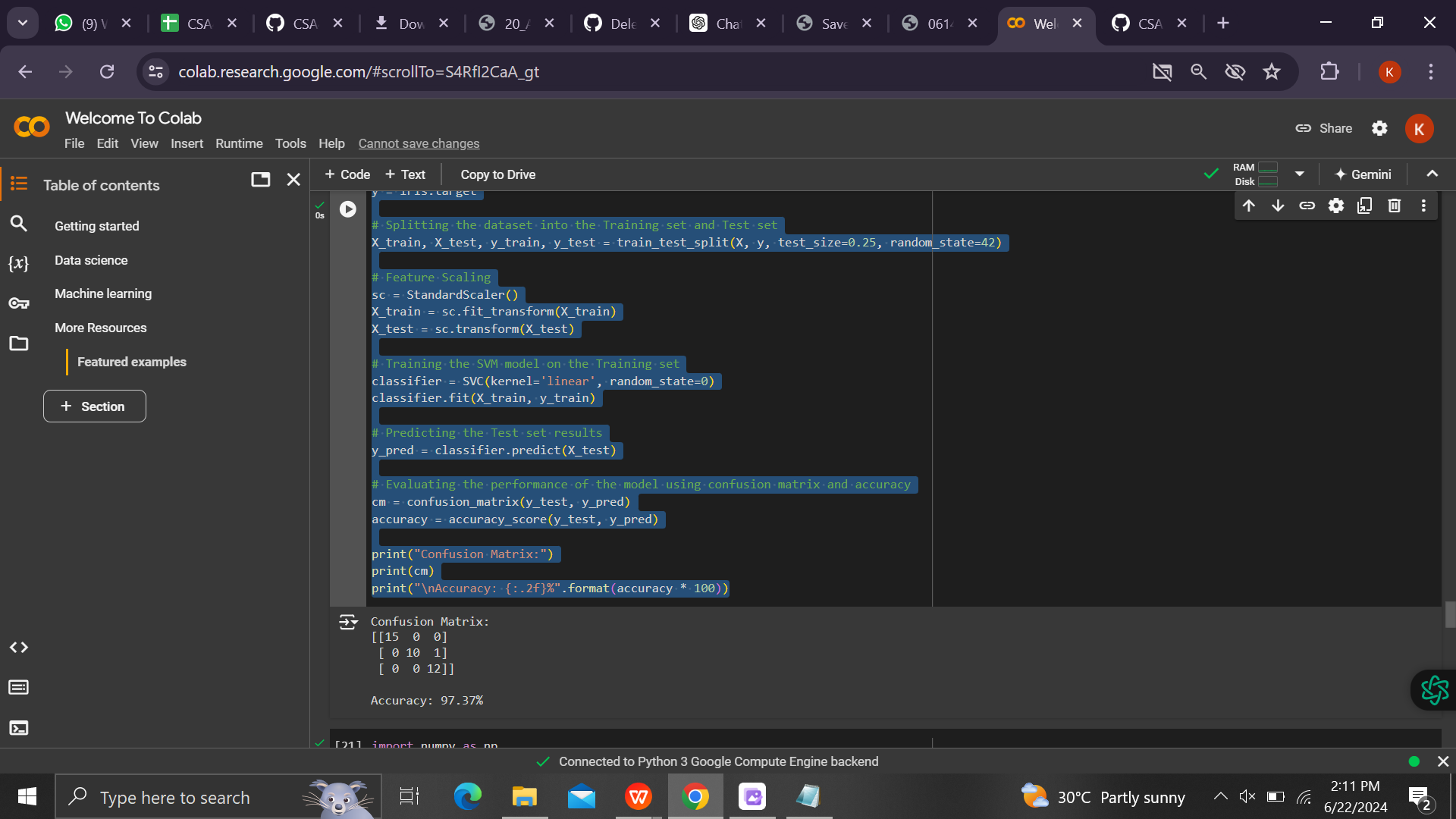
accuracy = accuracy\_score(y\_test, y\_pred)

print("Confusion Matrix:")

print(cm)

print("\nAccuracy: {:.2f}%".format(accuracy \* 100))

**Output:**

****

**6(F)**

**Program:**

import numpy as np

import pandas as pd

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import confusion\_matrix, accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Splitting the dataset into the Training set and Test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# Training the Random Forest model on the Training set

classifier = RandomForestClassifier(n\_estimators=100, random\_state=0)

classifier.fit(X\_train, y\_train)

# Predicting the Test set results

y\_pred = classifier.predict(X\_test)

# Evaluating the performance of the model using confusion matrix and accuracy

cm = confusion\_matrix(y\_test, y\_pred)

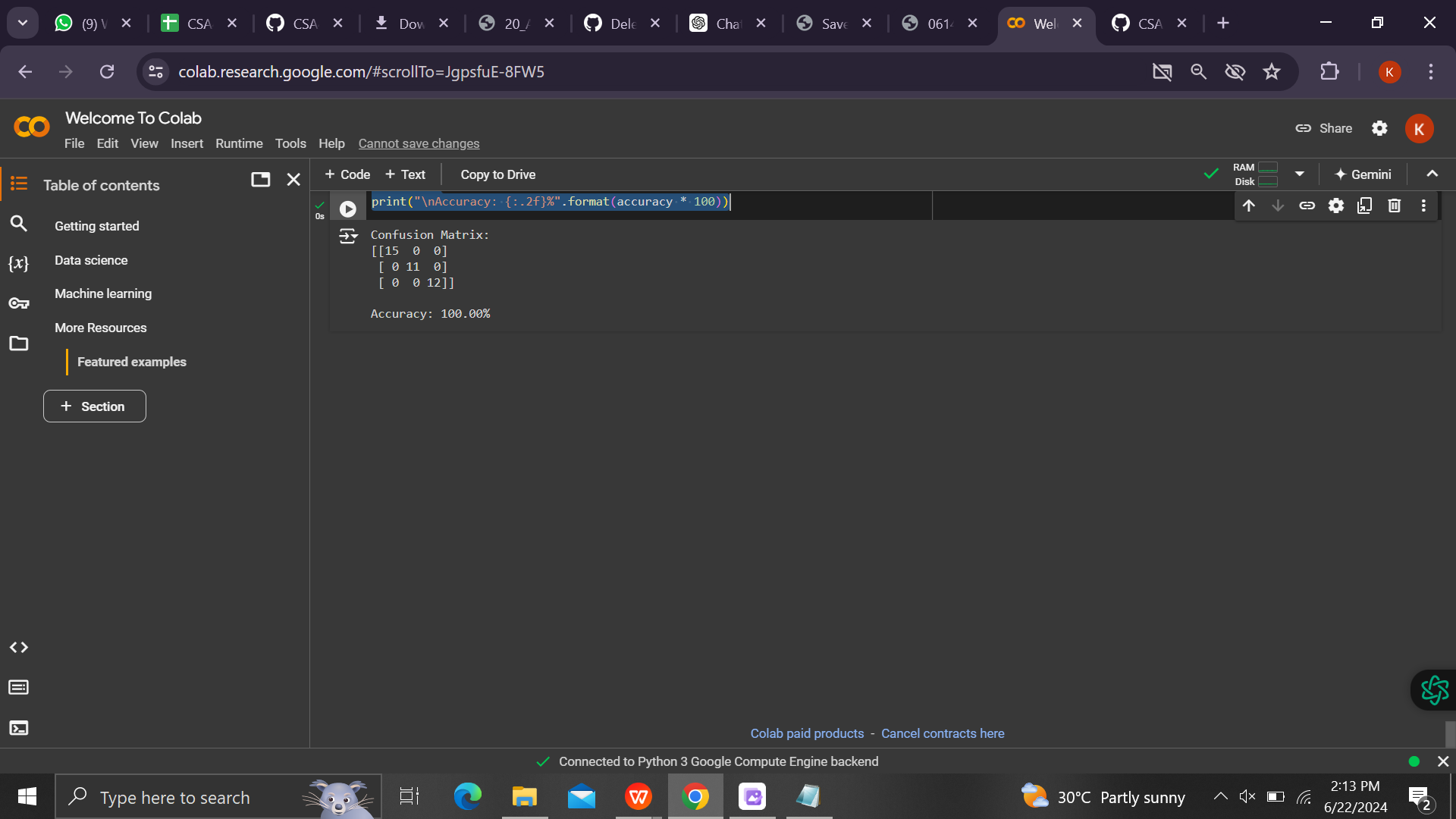
accuracy = accuracy\_score(y\_test, y\_pred)

print("Confusion Matrix:")

print(cm)

print("\nAccuracy: {:.2f}%".format(accuracy \* 100))

**Output:**

****