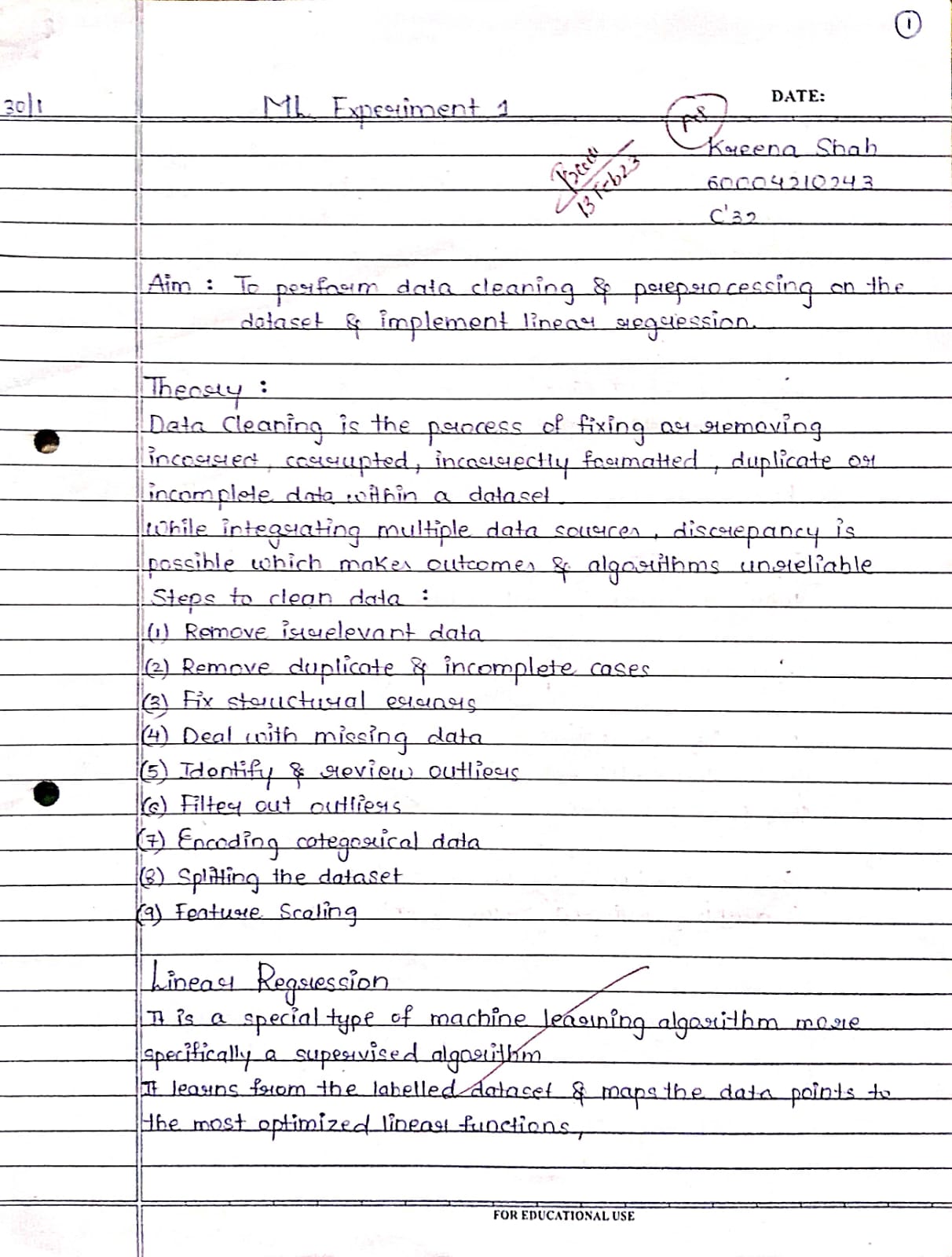
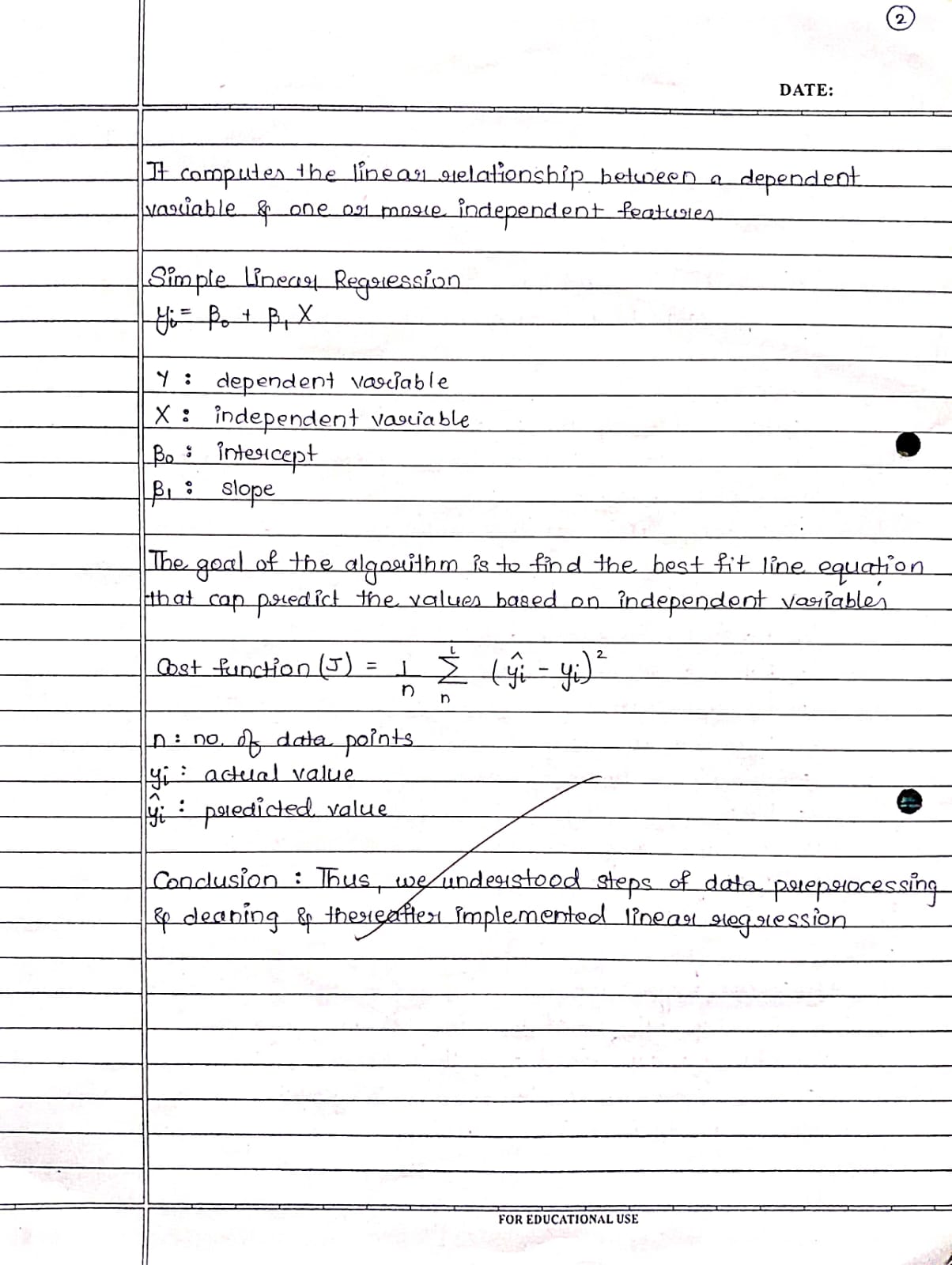
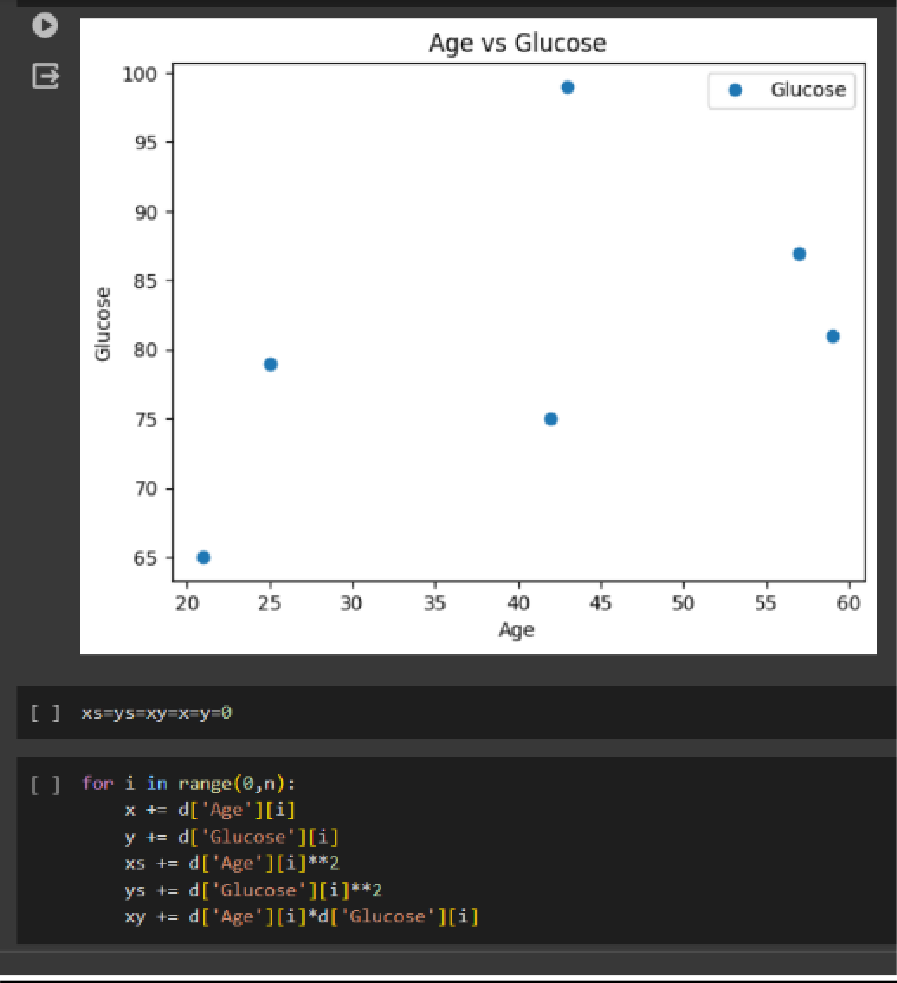
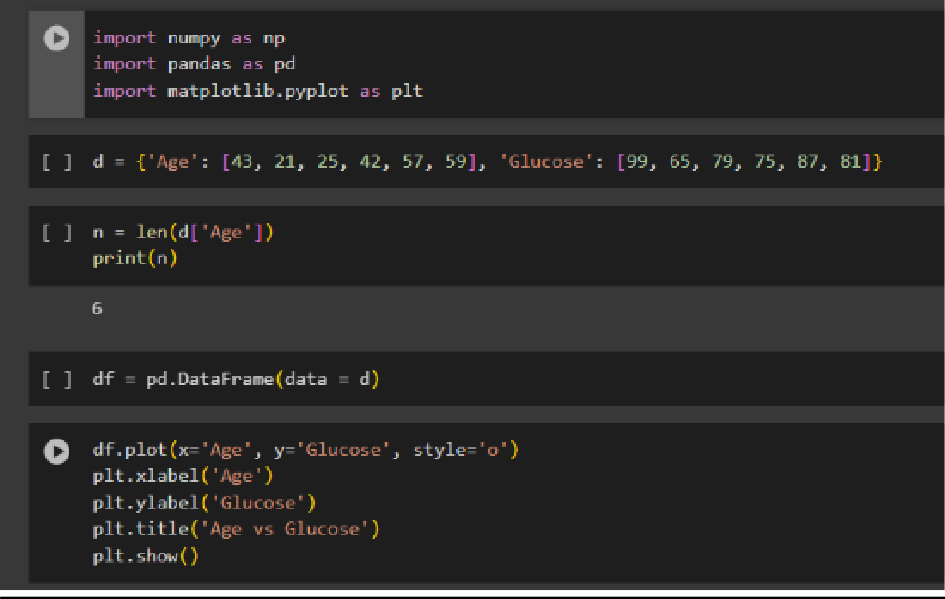
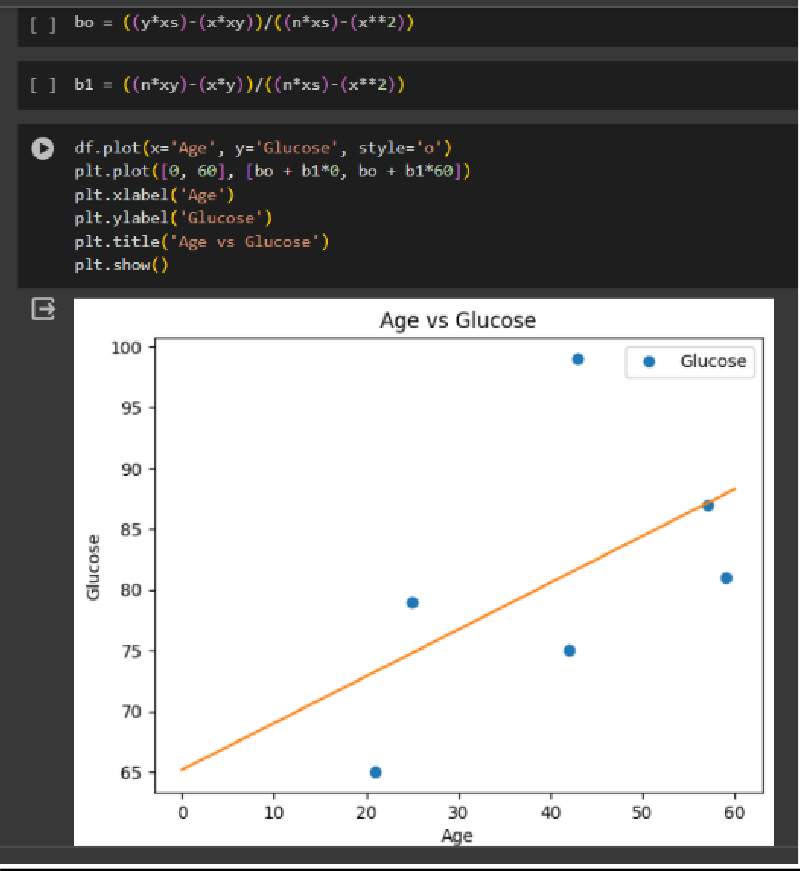
###### ML Experiment 1



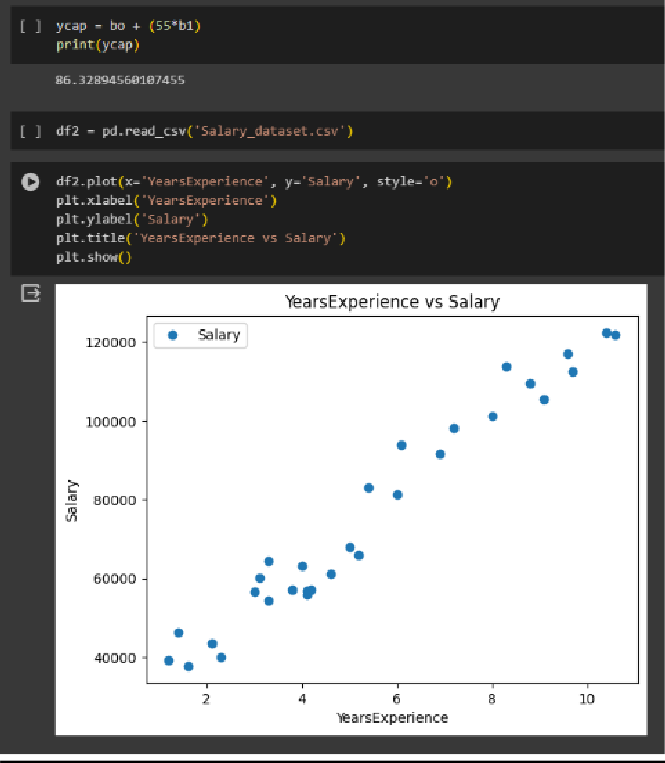


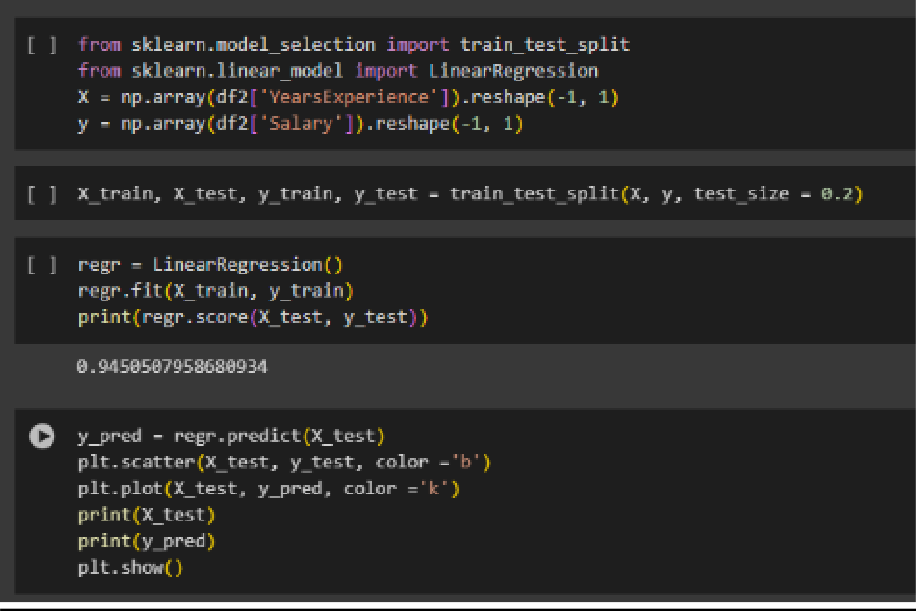
Using direct formulas.



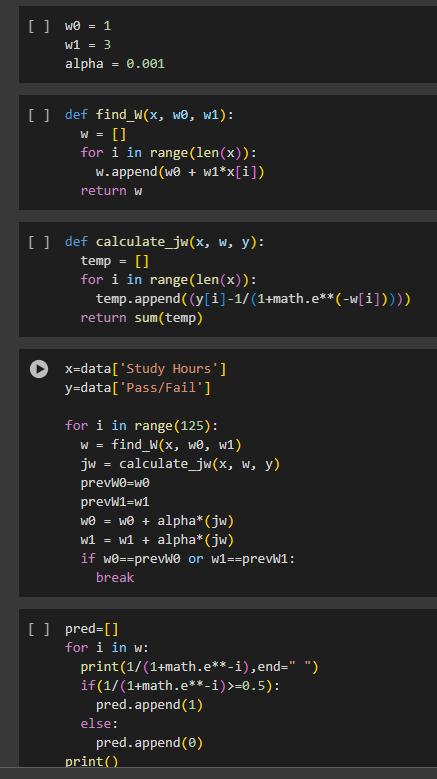
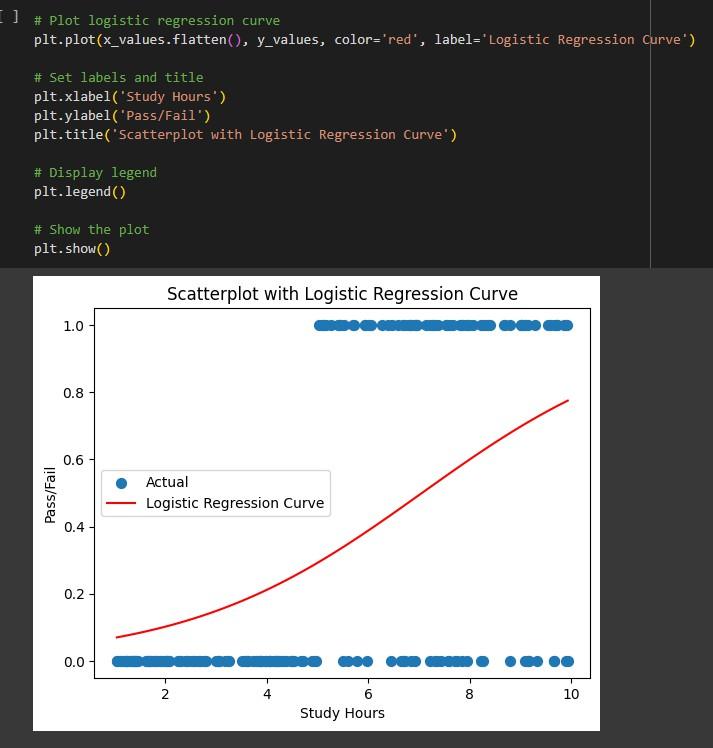


Then on an imported dataset using inbuilt functions and libraries

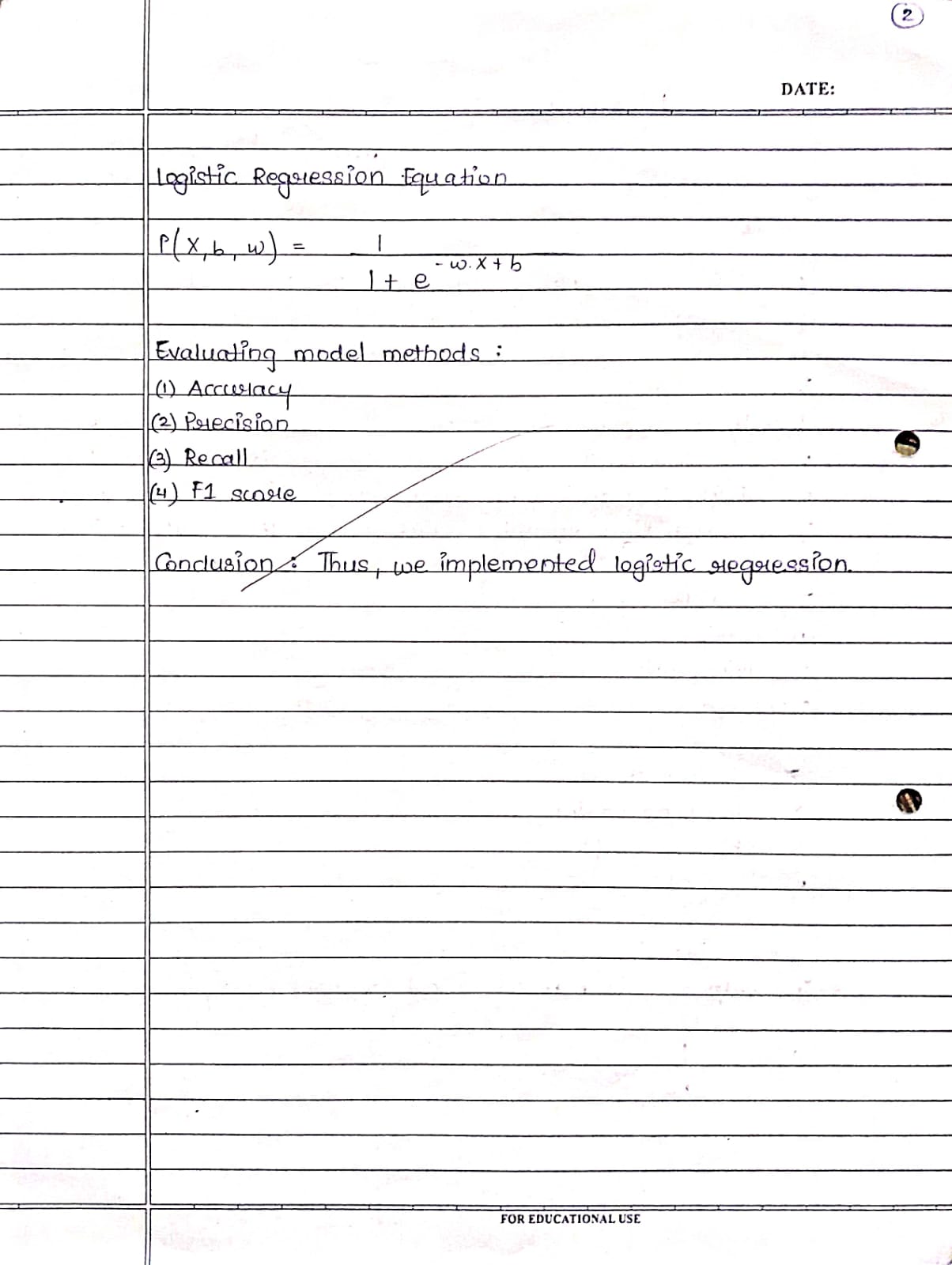
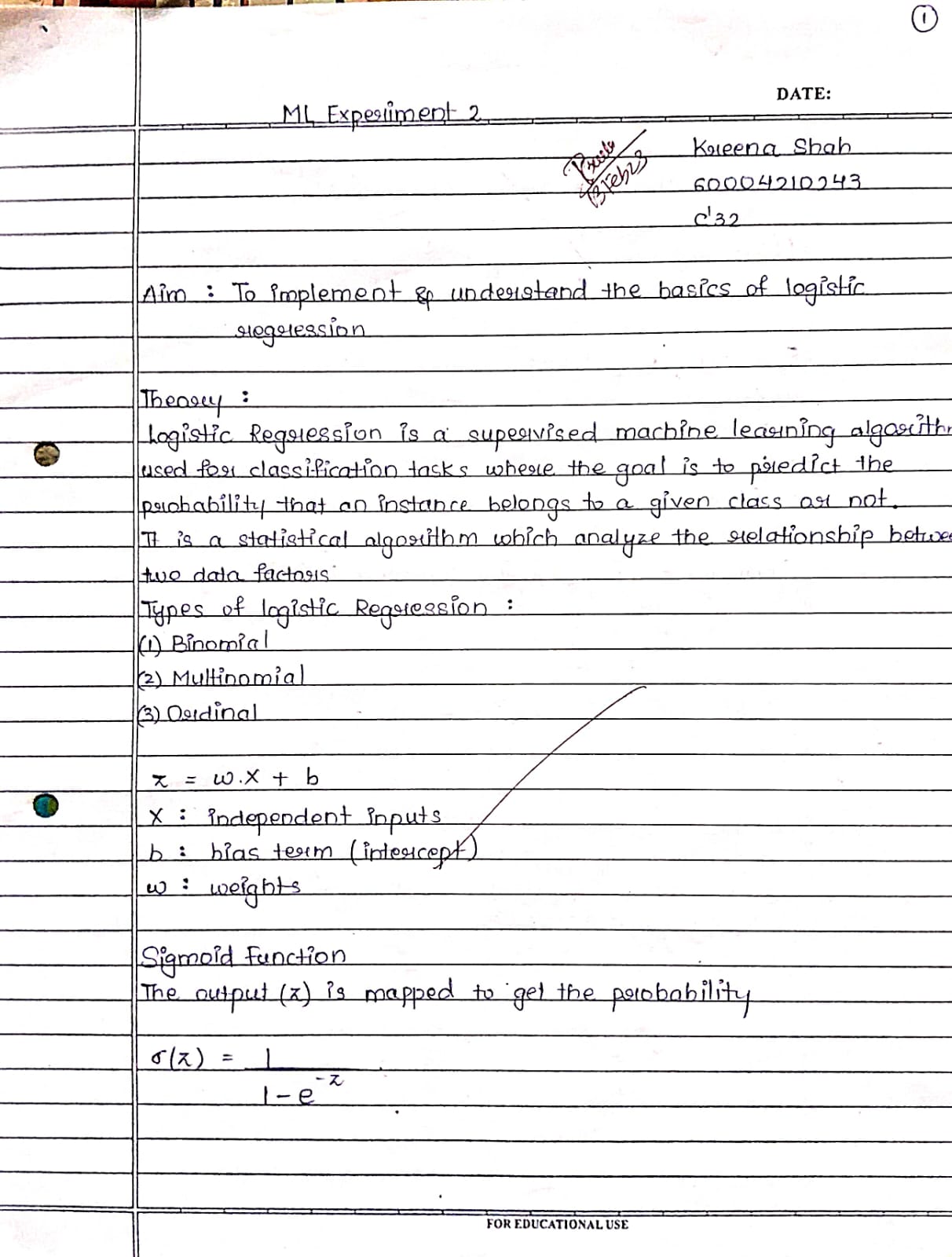




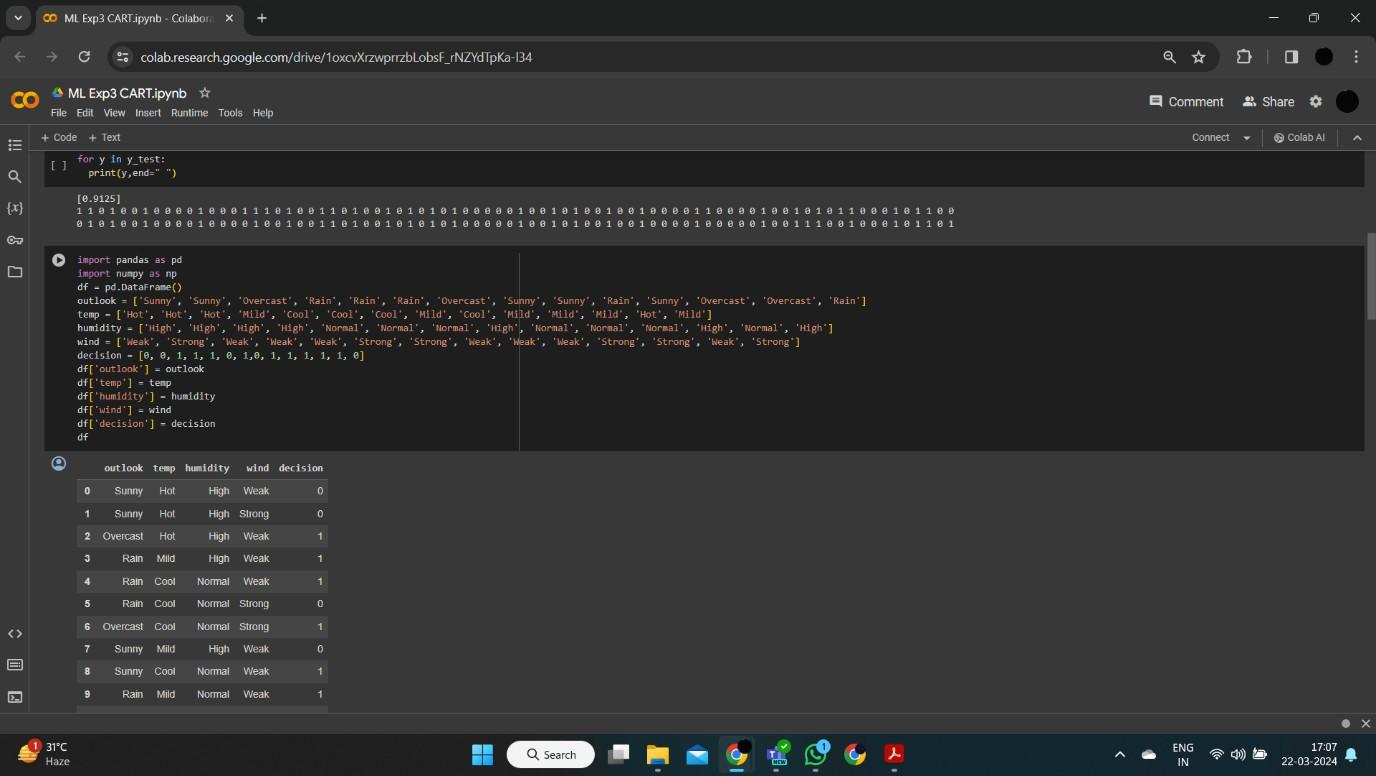
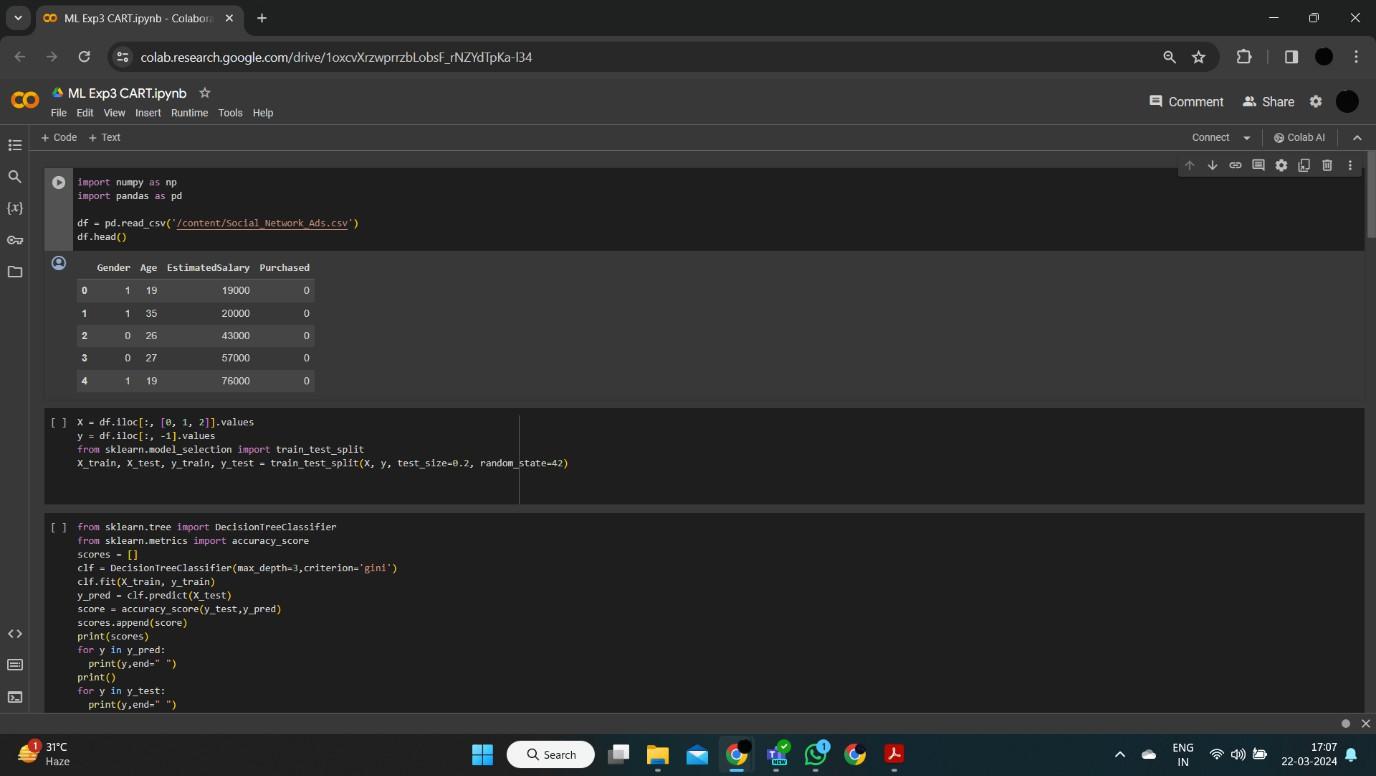
###### ML Experiment 2

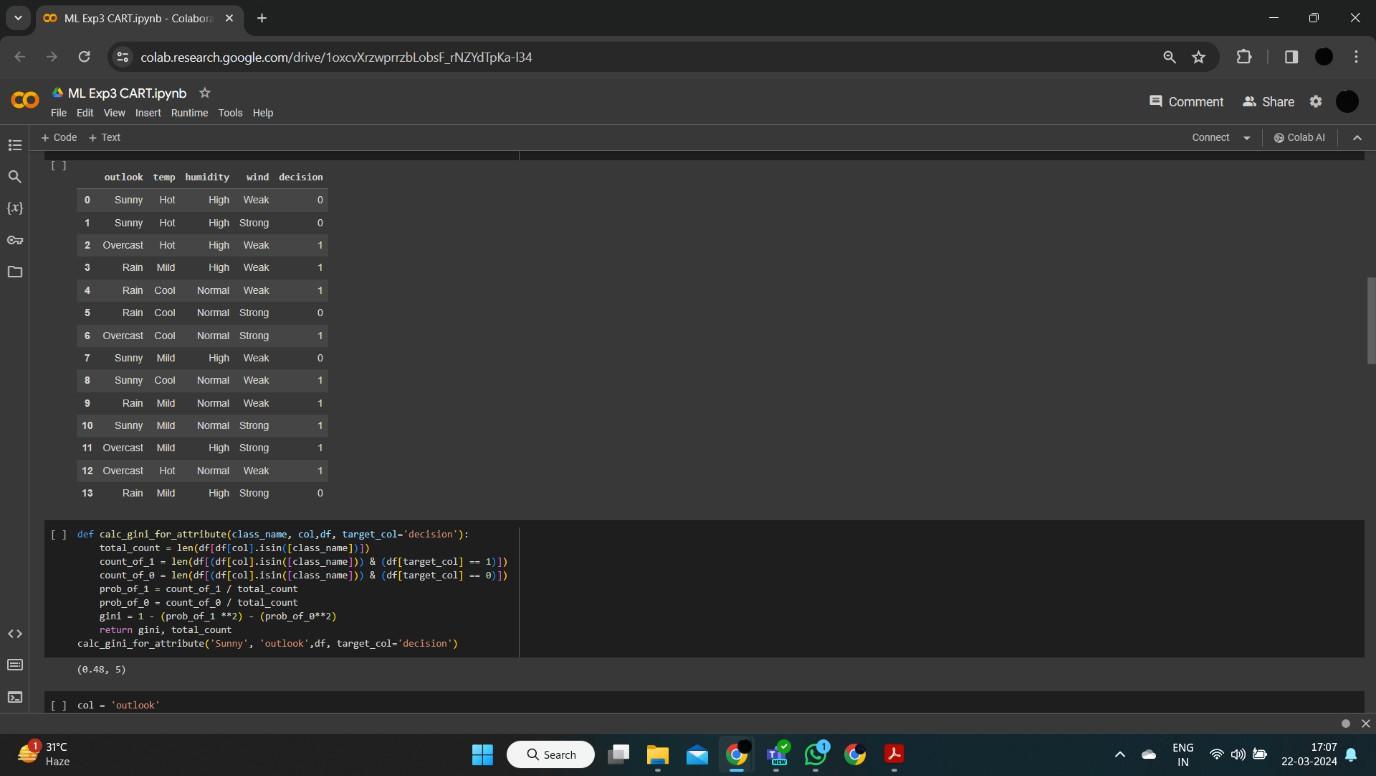
Implementation:

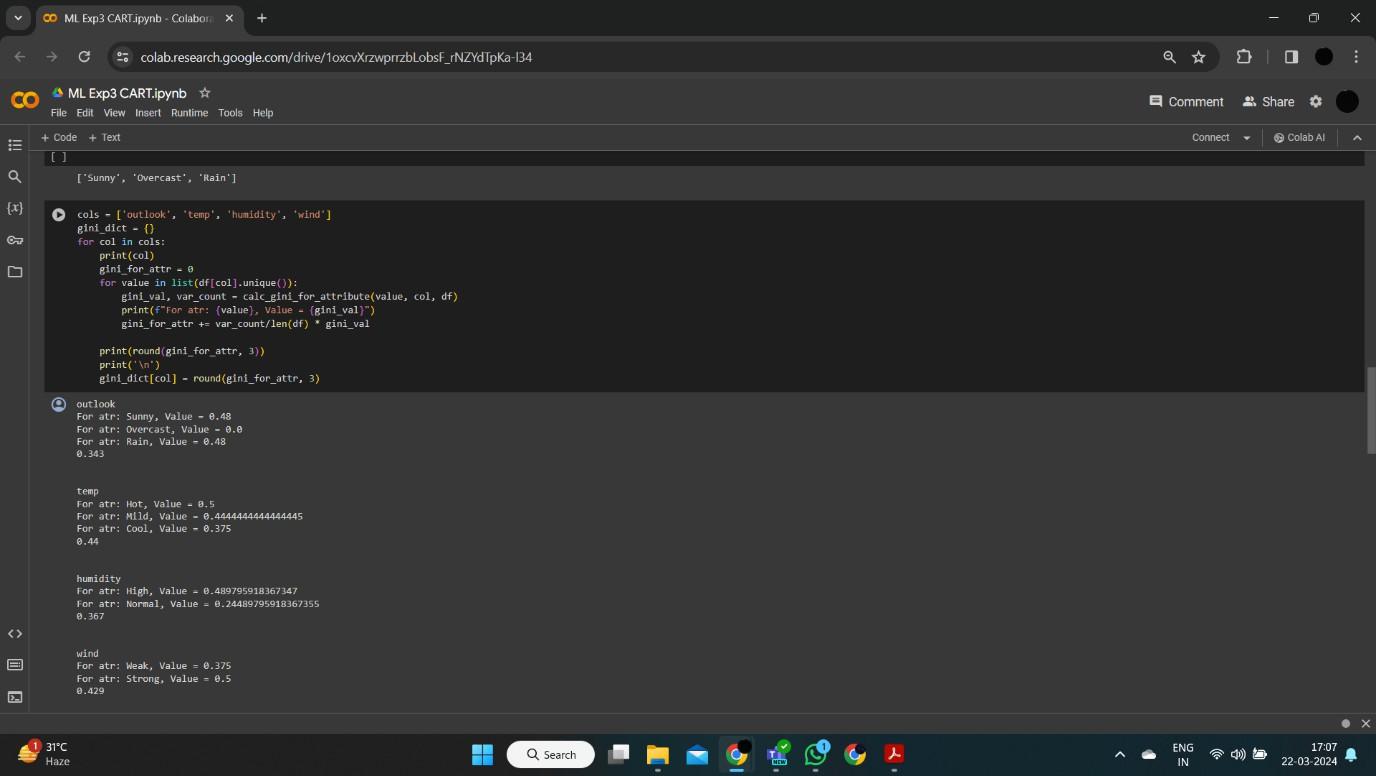


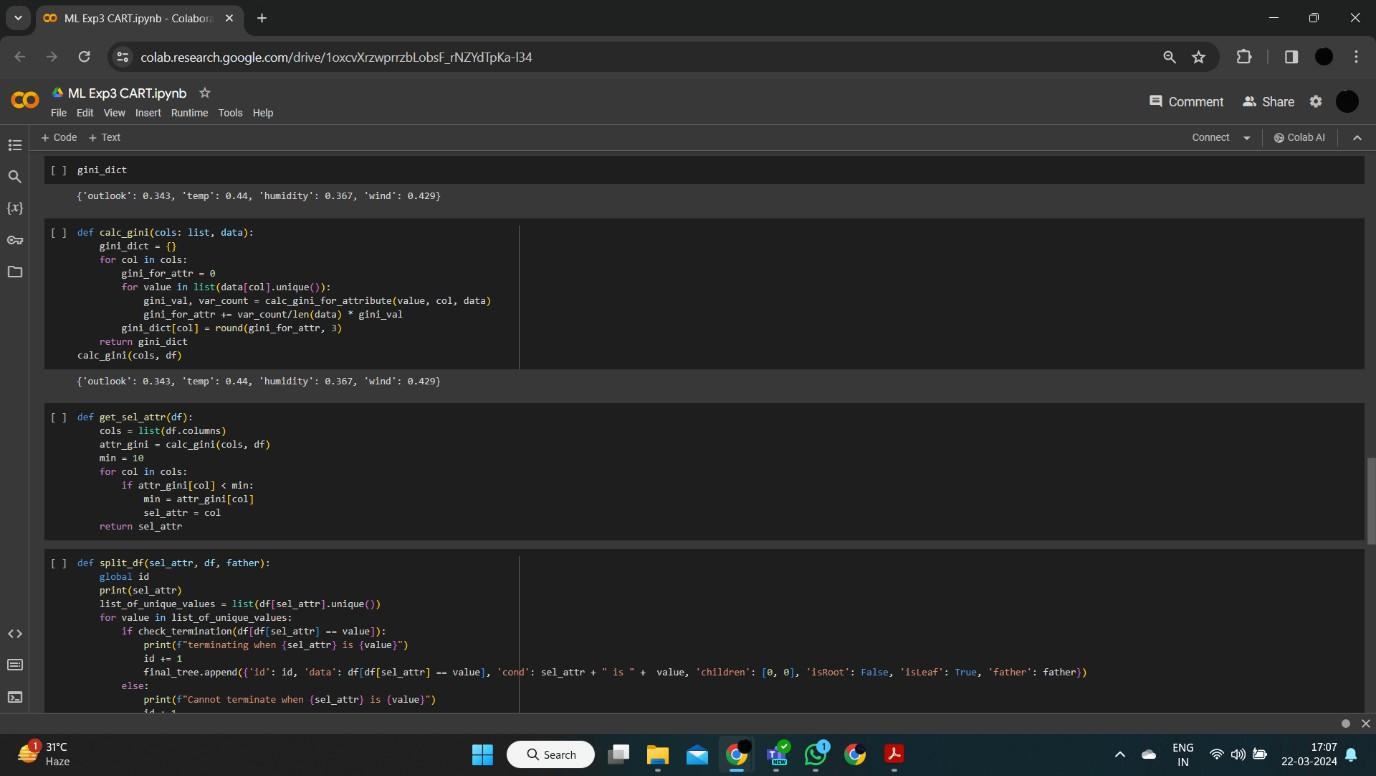


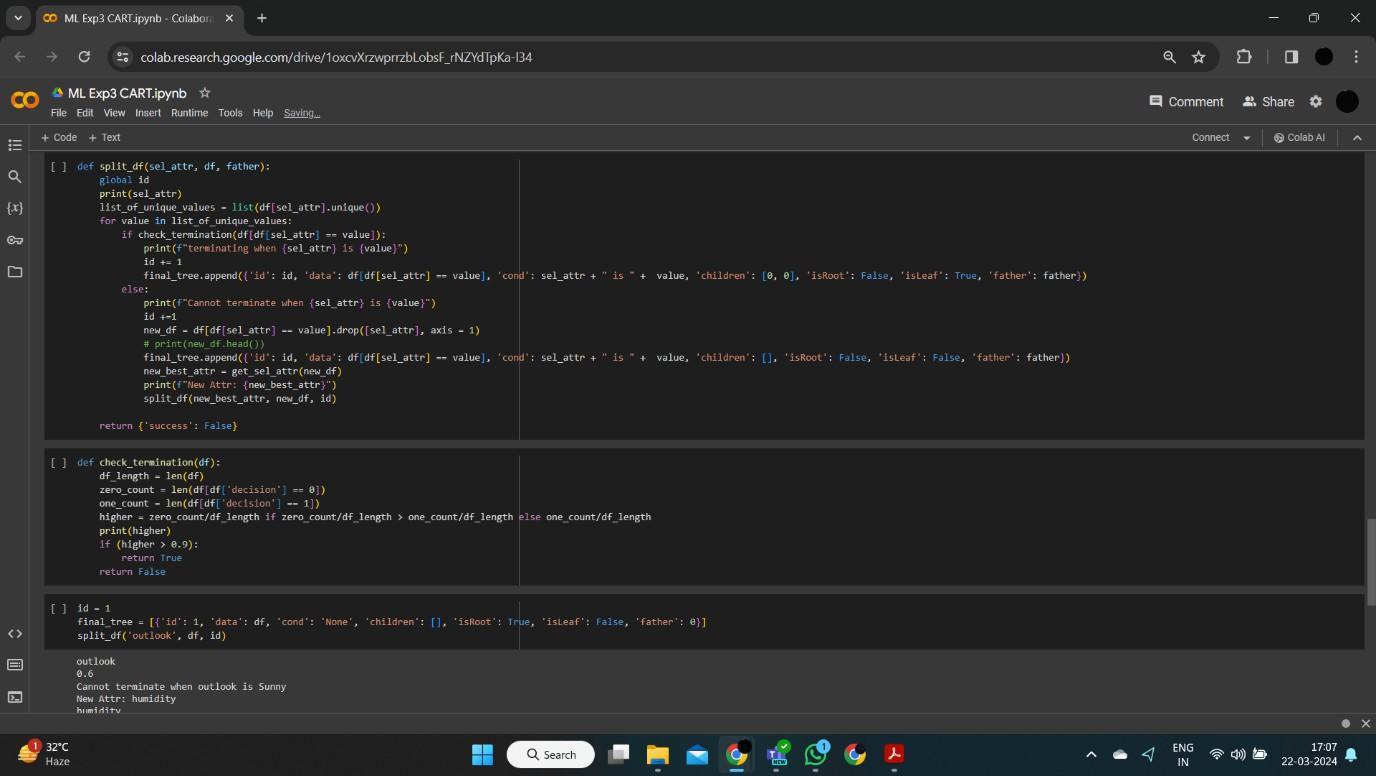
###### ML Experiment 3

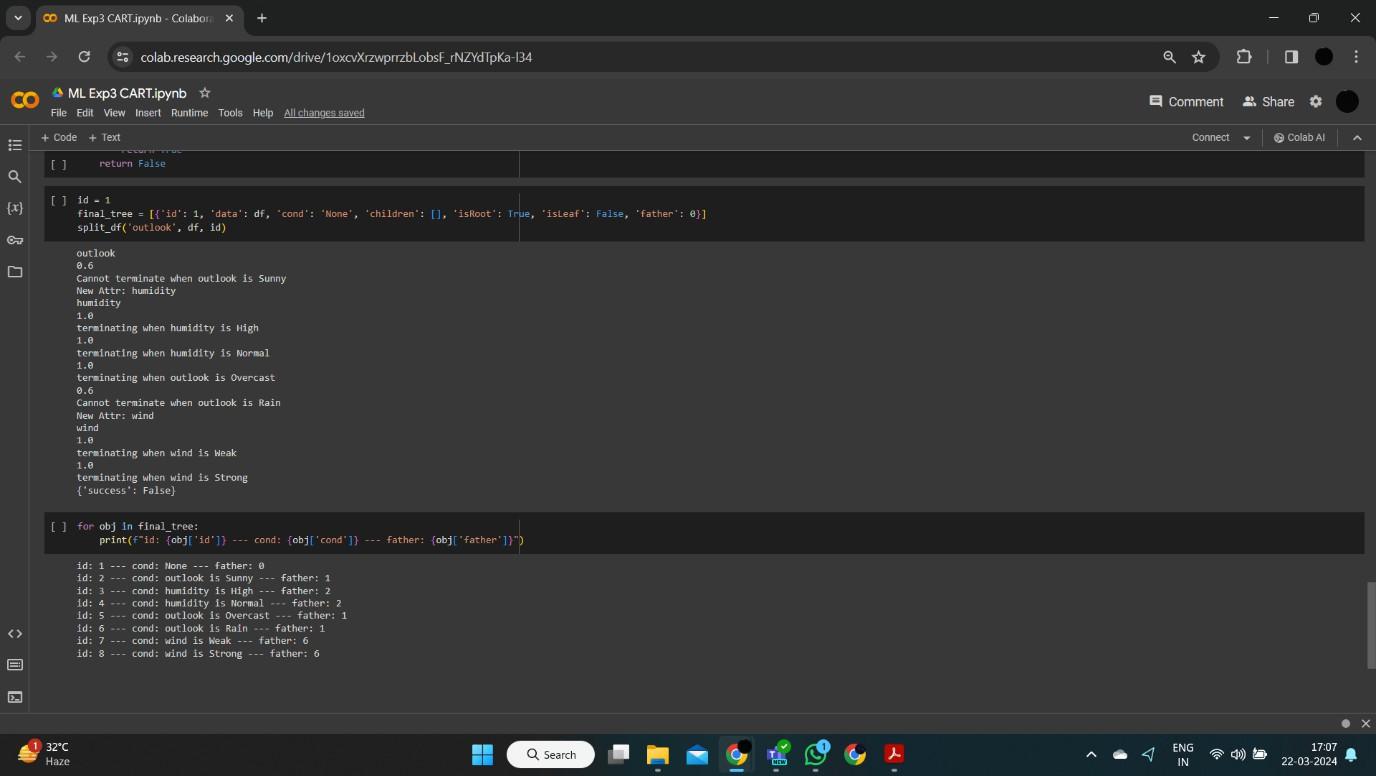


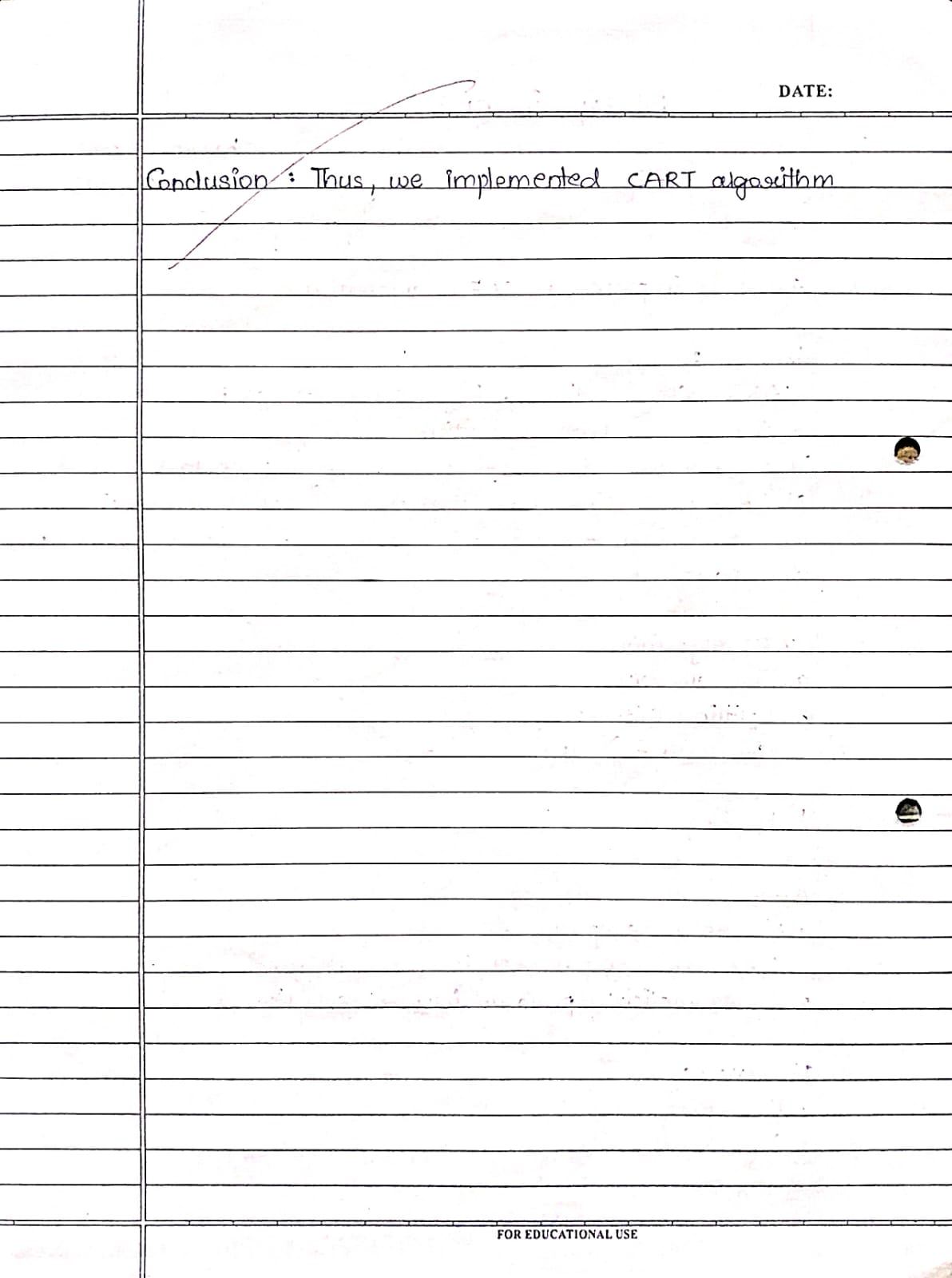
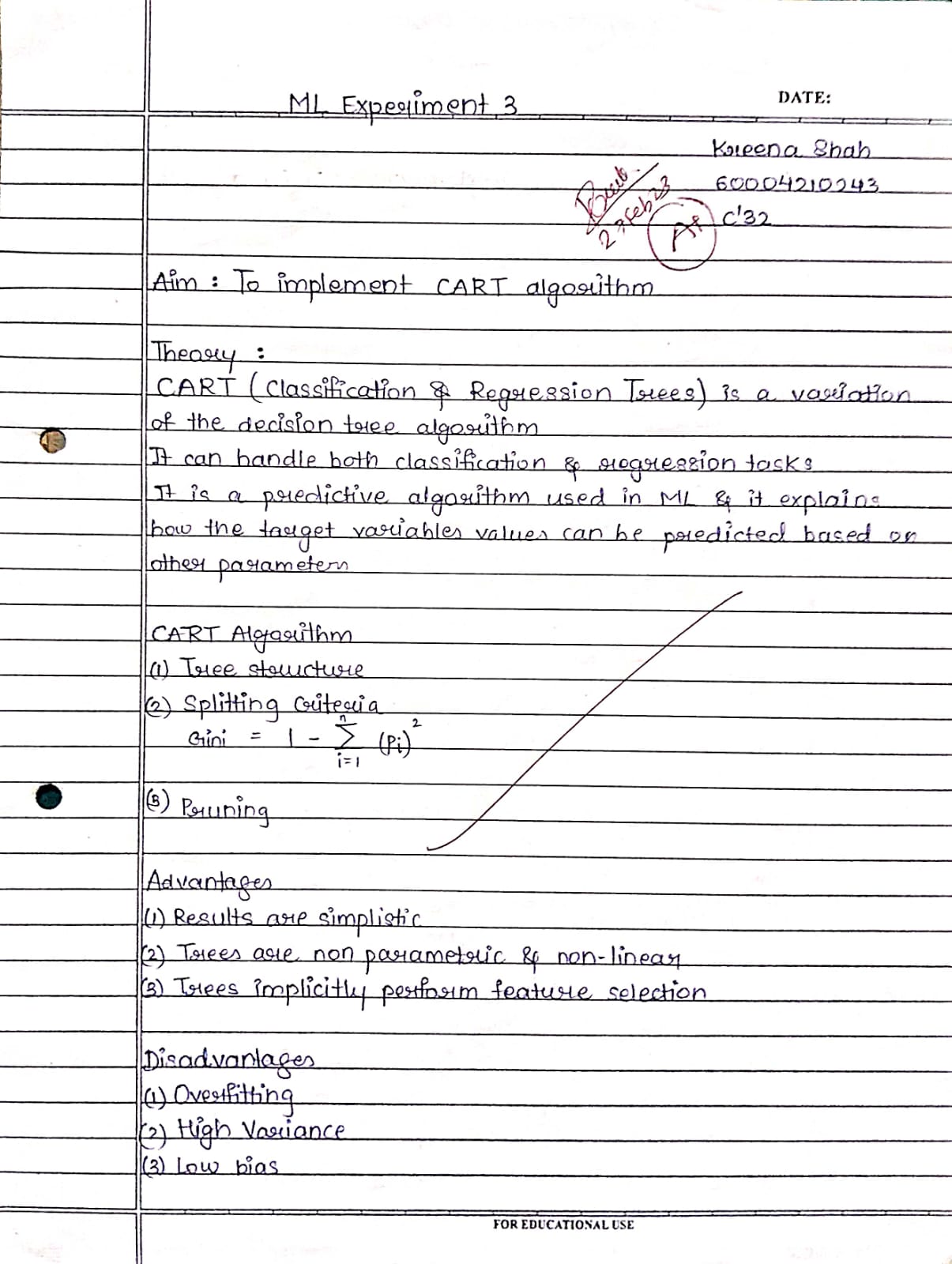




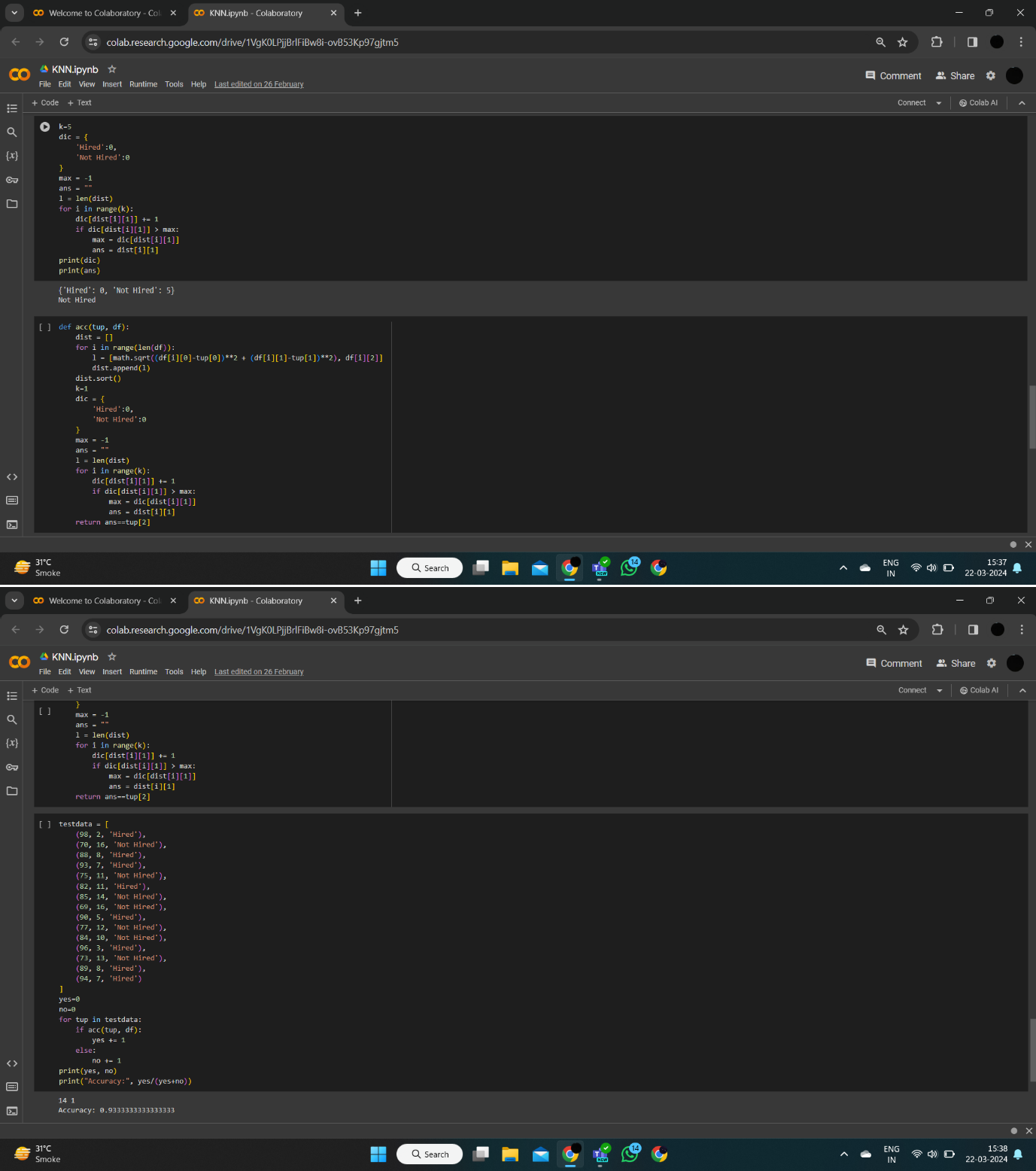


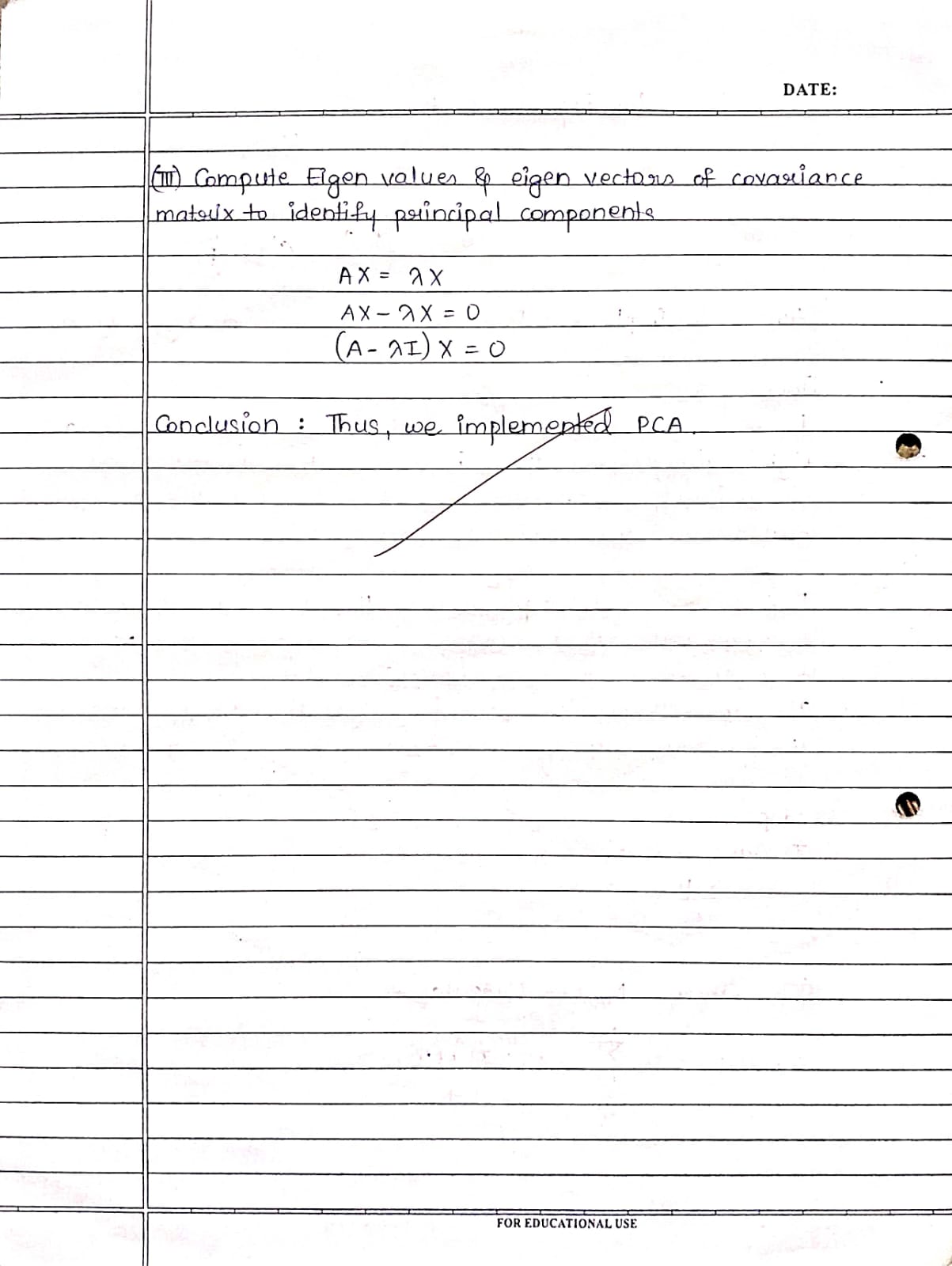
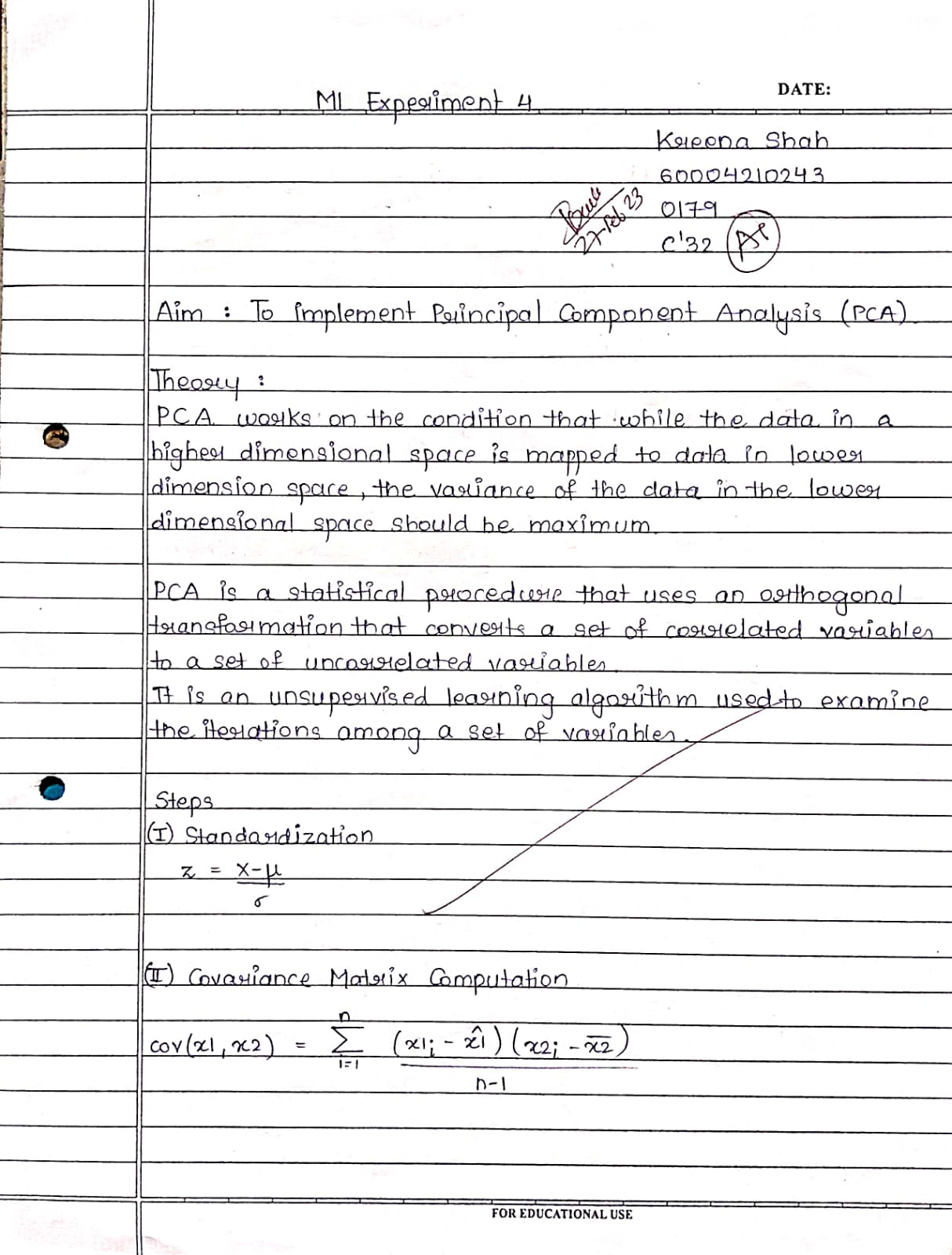




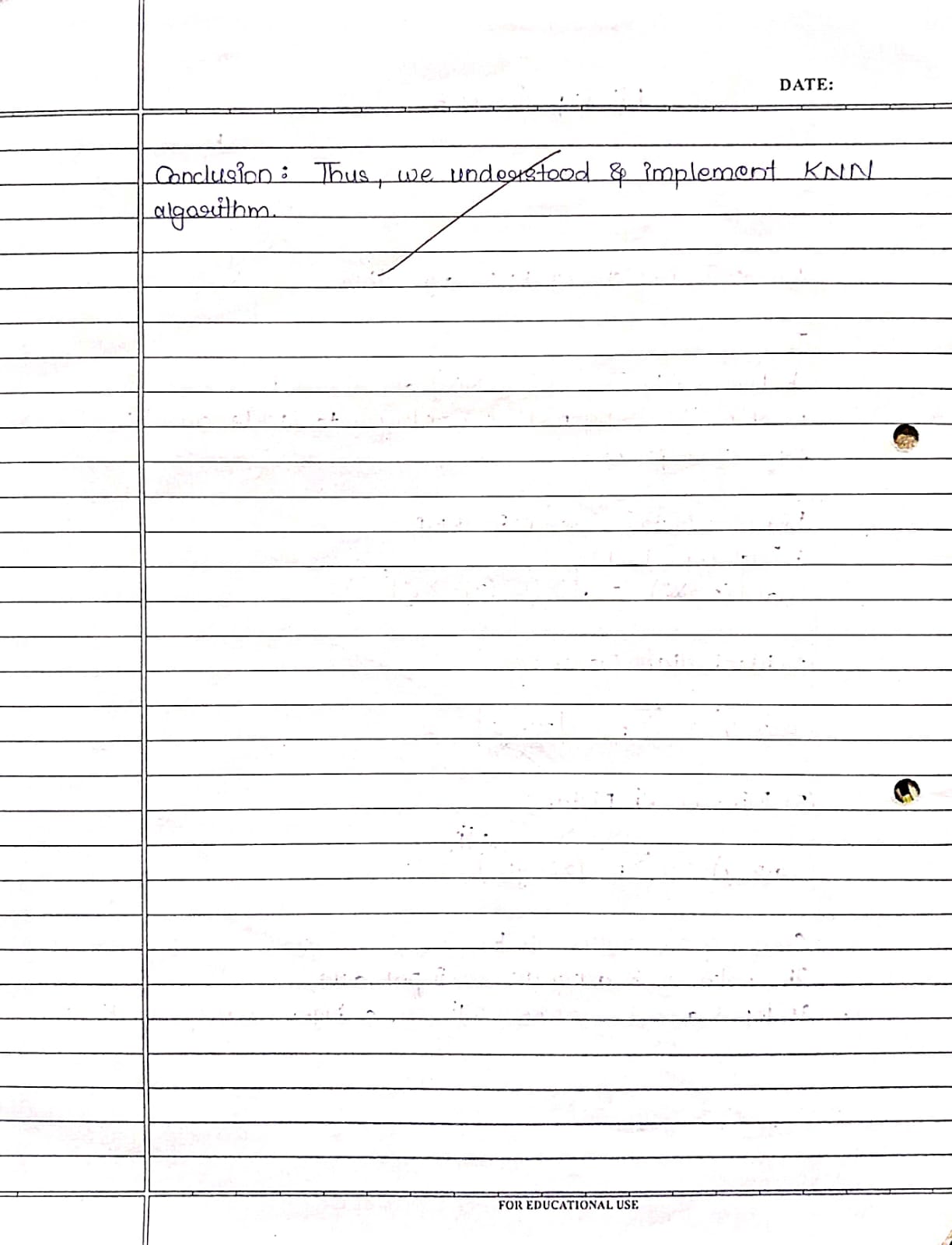
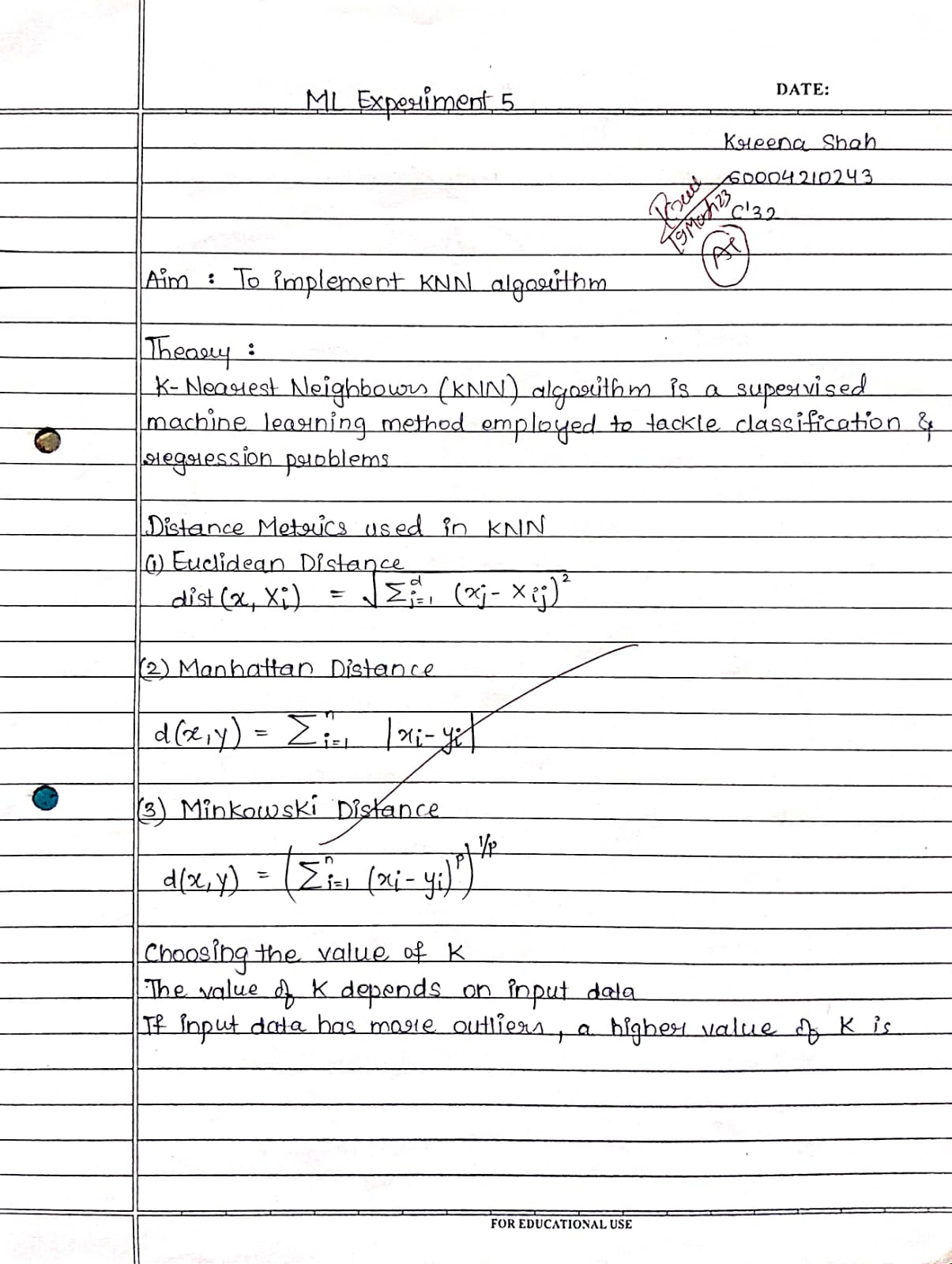


###### ML Experiment 4





###### ML Experiment 5



import numpy as np

import matplotlib.pyplot as plt import pandas as pd

from sklearn.datasets import load\_digits from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler digits\_data = load\_digits()

X = digits\_data.data scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X) pca = PCA()

X\_pca = pca.fit\_transform(X\_scaled) eigenvalues = pca.explained\_variance\_ plt.figure(figsize=(10, 6))

plt.plot(range(1, len(eigenvalues) + 1), eigenvalues, marker='o', linestyle='-')

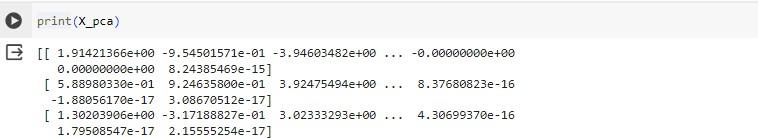
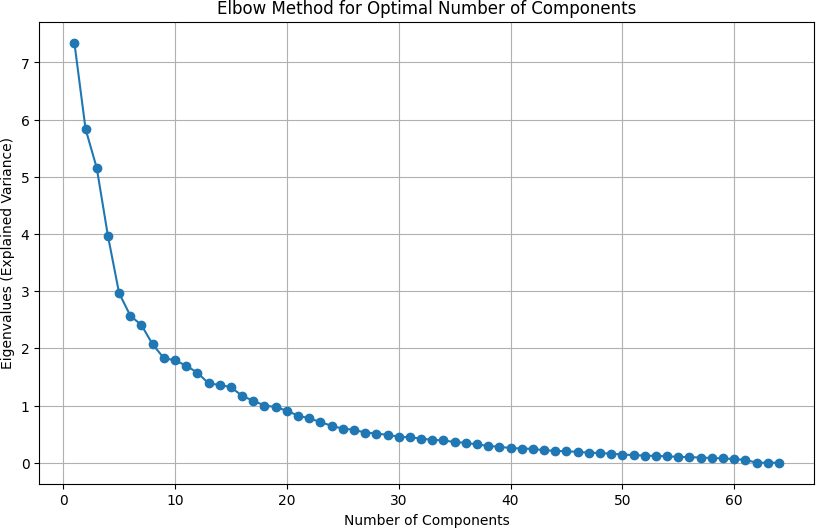
plt.title('Elbow Method for Optimal Number of Components') plt.xlabel('Number of Components') plt.ylabel('Eigenvalues (Explained Variance)') plt.grid(True)

plt.show() optimal\_num\_components = 10

X\_reduced = X\_pca[:, :optimal\_num\_components]

df\_reduced = pd.DataFrame(X\_reduced, columns=[f'PC{i}' for i in range(1, optimal\_num\_components + 1)])

df\_reduced['target'] = digits\_data.target df\_reduced.to\_csv('reduced\_digits\_dataset.csv', index=False) print("Digits Wine dataset saved successfully.")



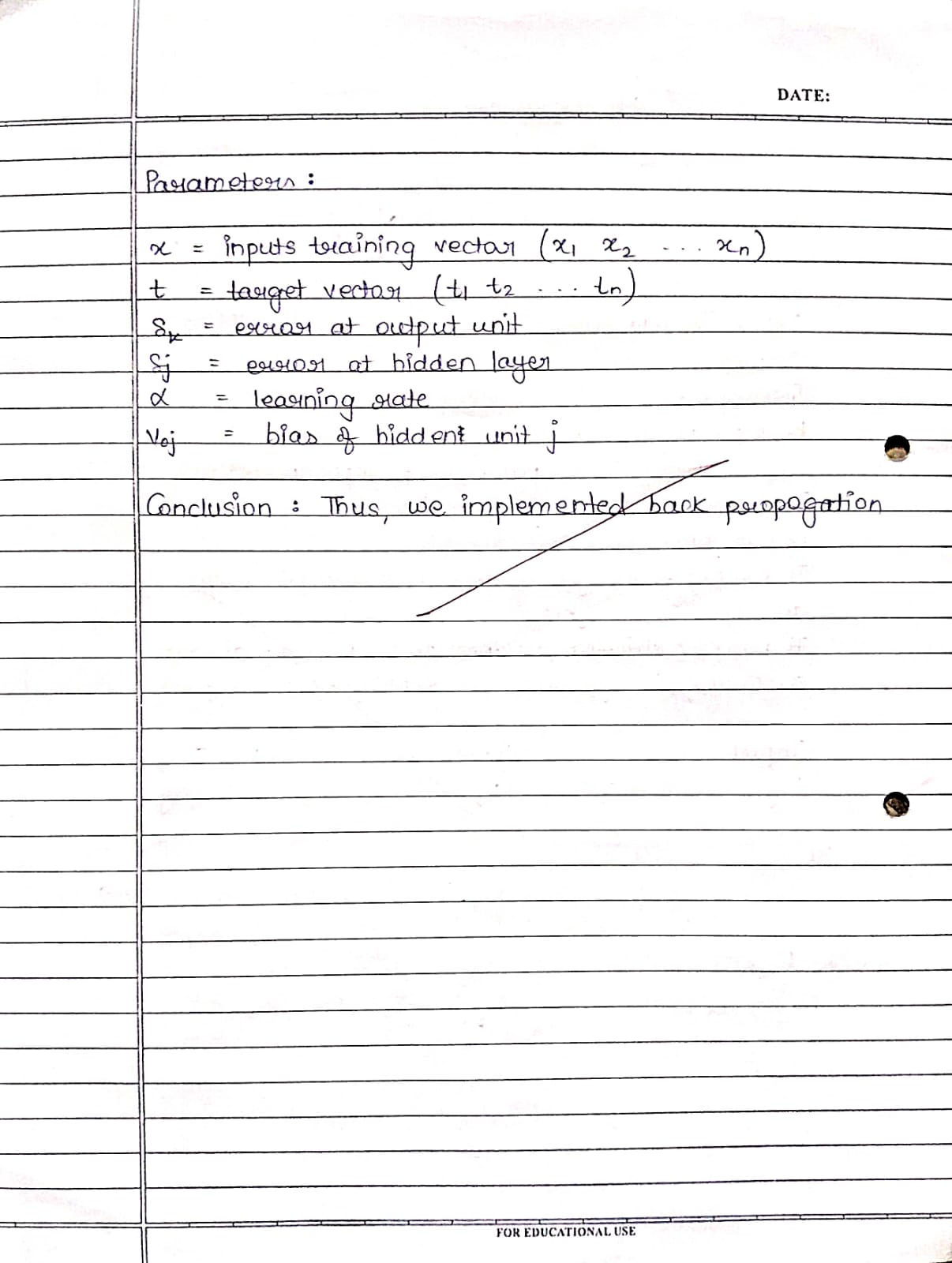
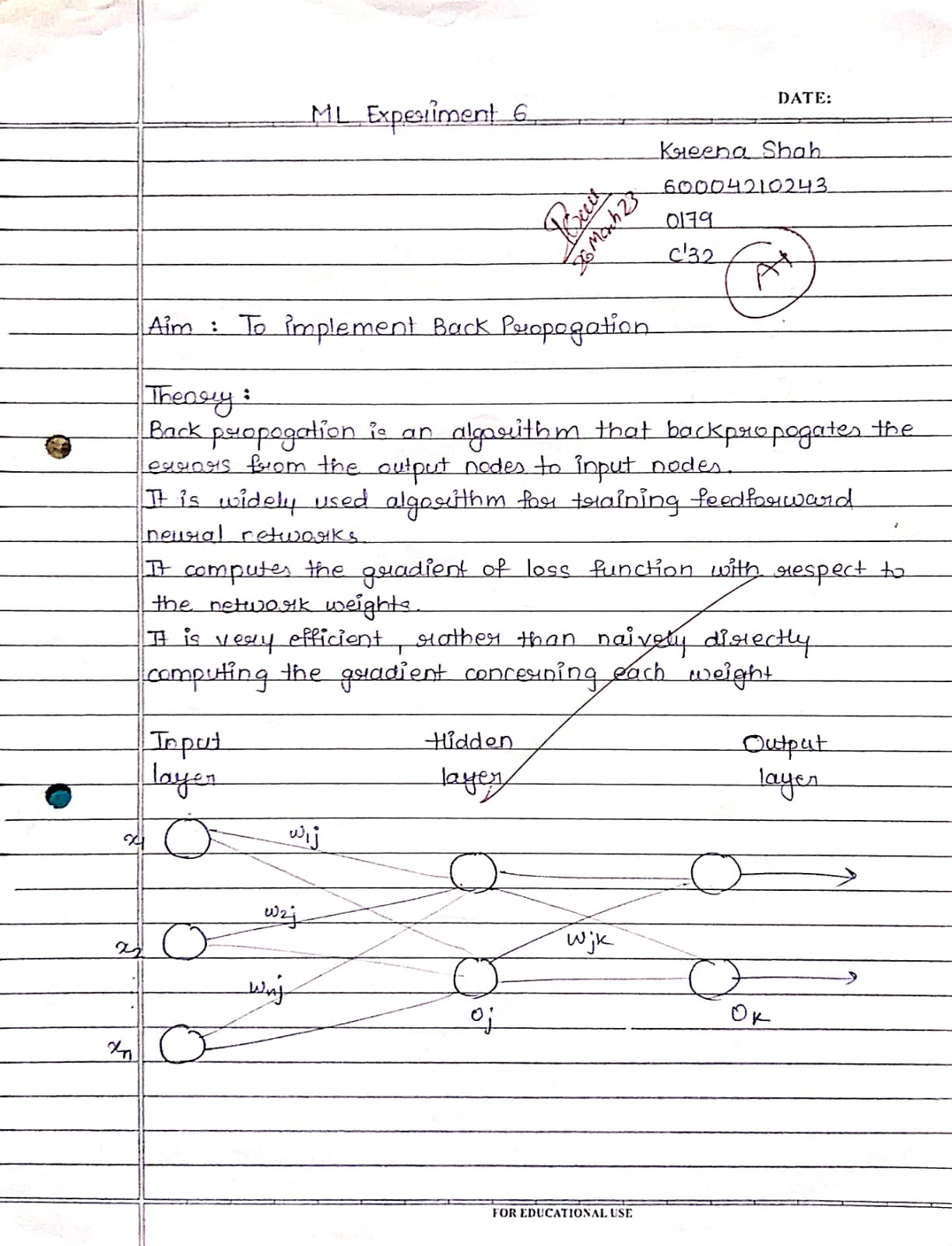
###### 

###### 

###### 

###### 

###### ML Experiment 6



import numpy as np class NeuralNetwork:

def init (self, input\_size, hidden\_size, output\_size):

self.input\_size = input\_size self.hidden\_size = hidden\_size self.output\_size = output\_size # Initialize weights and biases

self.weights\_input\_hidden = np.random.randn(self.input\_size, self.hidden\_size)

self.bias\_input\_hidden = np.random.randn(1, self.hidden\_size) self.weights\_hidden\_output = np.random.randn(self.hidden\_size, self.output\_size)

self.bias\_hidden\_output = np.random.randn(1, self.output\_size) def sigmoid(self, x):

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

def sigmoid\_derivative(self, x):

return x \* (1 - x)

def forward(self, inputs):

self.hidden\_input = np.dot(inputs, self.weights\_input\_hidden) + self.bias\_input\_hidden

self.hidden\_output = self.sigmoid(self.hidden\_input) self.final\_input = np.dot(self.hidden\_output,

self.weights\_hidden\_output) + self.bias\_hidden\_output self.final\_output = self.sigmoid(self.final\_input)

return self.hidden\_output, self.final\_output

def backward(self, inputs, targets, learning\_rate): error = targets - self.final\_output

delta\_output = error \* self.sigmoid\_derivative(self.final\_output) delta\_hidden = np.dot(delta\_output, self.weights\_hidden\_output.T) \* self.sigmoid\_derivative(self.hidden\_output)

self.weights\_hidden\_output += np.dot(self.hidden\_output.T, delta\_output) \* learning\_rate

self.bias\_hidden\_output += np.sum(delta\_output, axis=0, keepdims=True) \* learning\_rate

self.weights\_input\_hidden += np.dot(inputs.T, delta\_hidden) \* learning\_rate

self.bias\_input\_hidden += np.sum(delta\_hidden, axis=0, keepdims=True) \* learning\_rate

return error

def train(self, inputs, targets, learning\_rate): hidden\_output, final\_output = self.forward(inputs) error = self.backward(inputs, targets, learning\_rate) print("Output of hidden layer:") print(hidden\_output)

print("Output of output layer:")

print(final\_output) print("Error found:") print(error)

print("Updated weights after 1 iteration:") print("Weights from input to hidden layer:") print(self.weights\_input\_hidden)

print("Weights from hidden to output layer:") print(self.weights\_hidden\_output)

dataset = pd.read\_csv('reduced\_digits\_dataset.csv') inputs = dataset.drop(columns=['target']).values

targets = dataset['target'].values.reshape(-1, 1) input\_size = inputs.shape[1]

output\_size = len(np.unique(targets)) hidden\_size = 3

nn = NeuralNetwork(input\_size, hidden\_size, output\_size) nn.train(inputs, targets, learning\_rate=0.1)

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

###### 

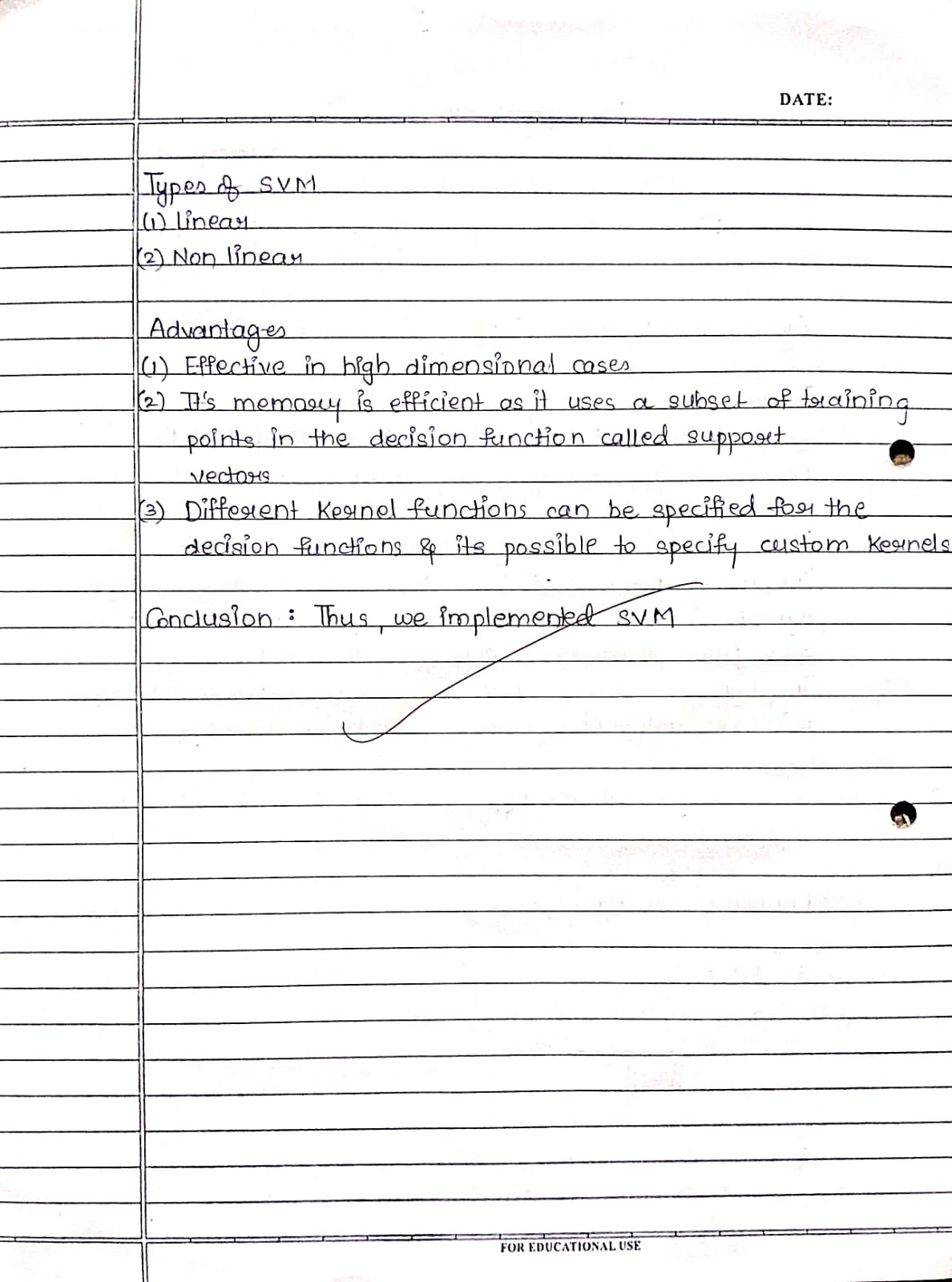
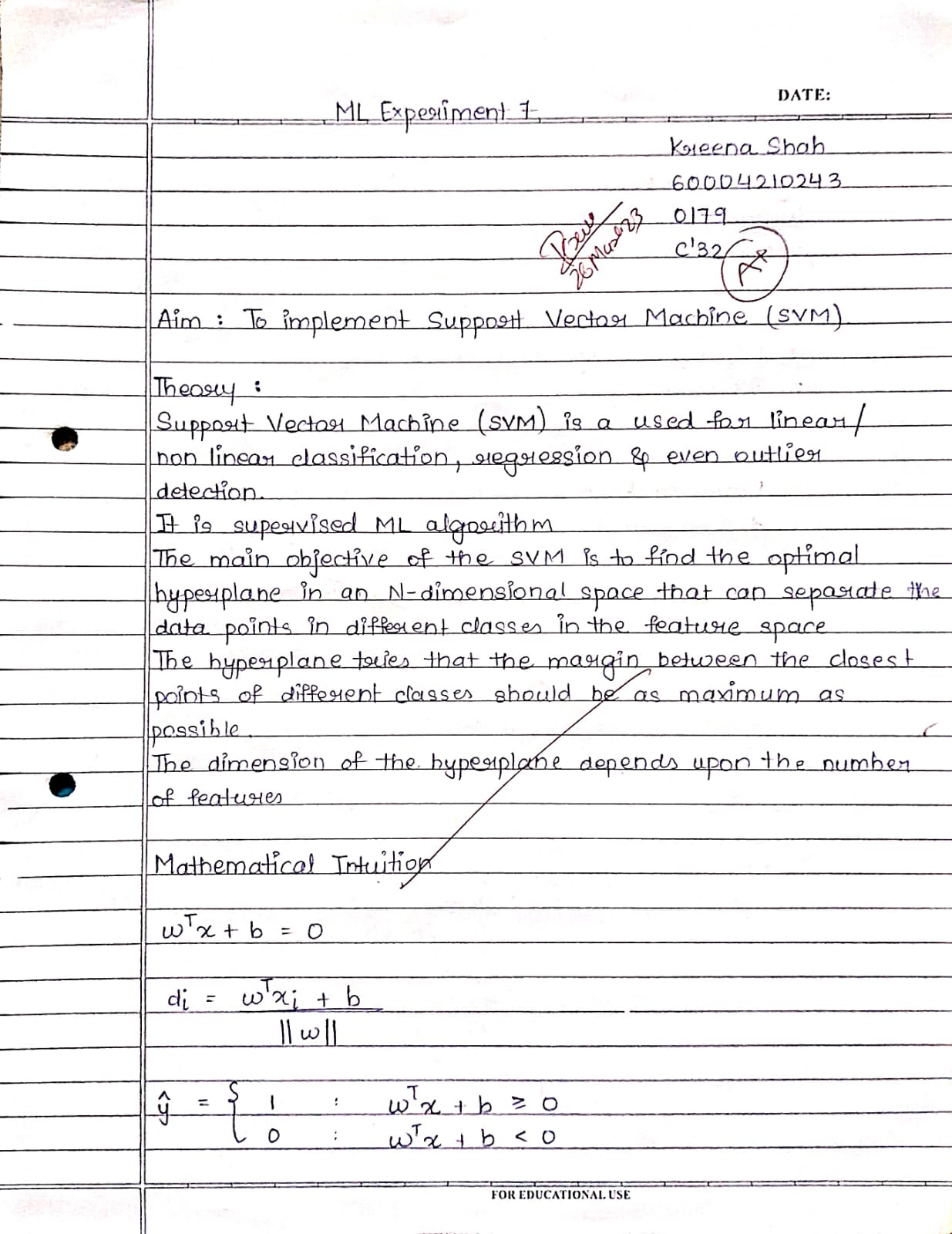
###### 

###### 

###### 

###### 

###### ML Experiment 7



from sklearn.datasets import load\_breast\_cancer from sklearn.model\_selection import train\_test\_split from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score data = load\_breast\_cancer()

X = data.data y = data.target X

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

classifier = GaussianNB() classifier.fit(X\_train, y\_train) y\_pred = classifier.predict(X\_test)

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

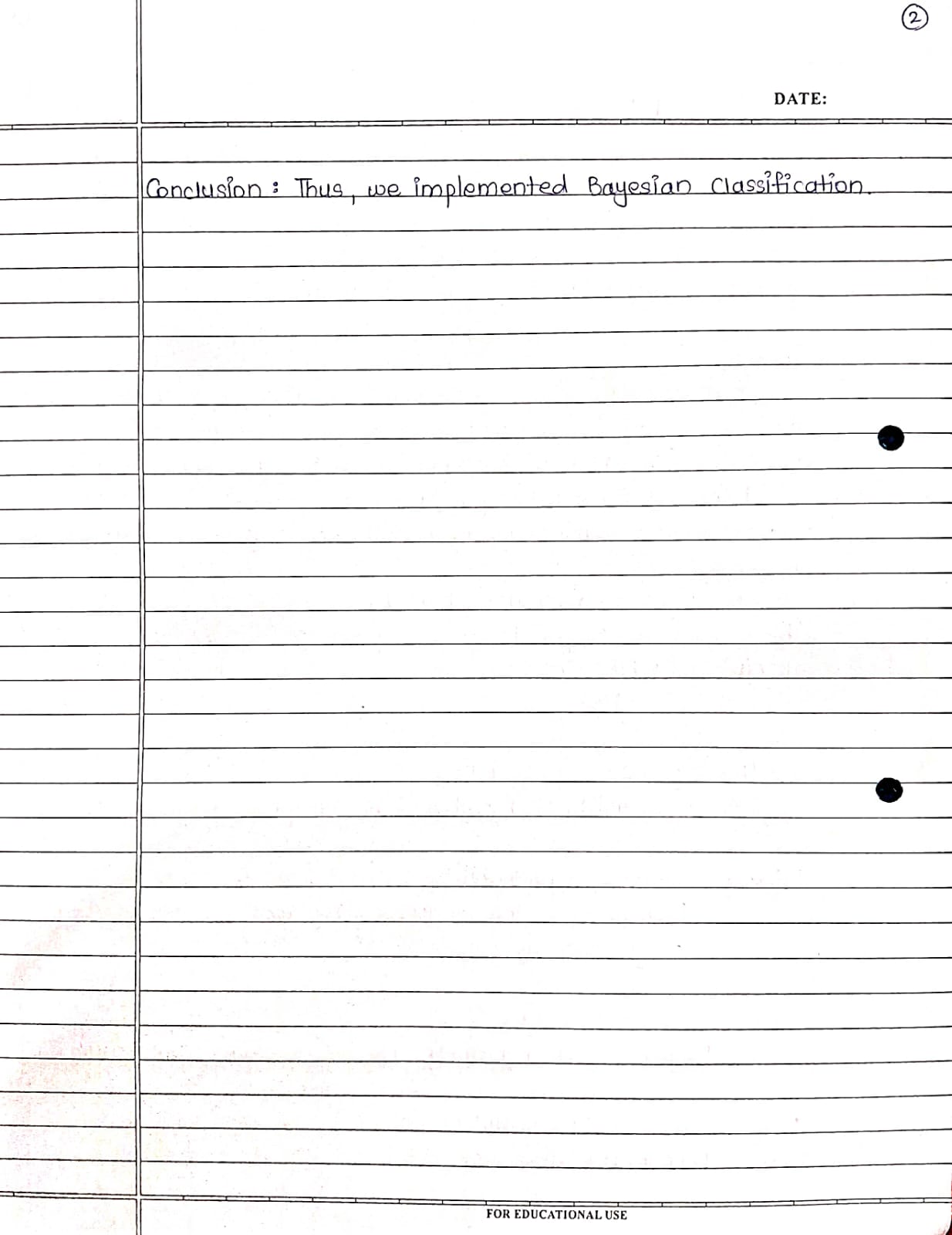
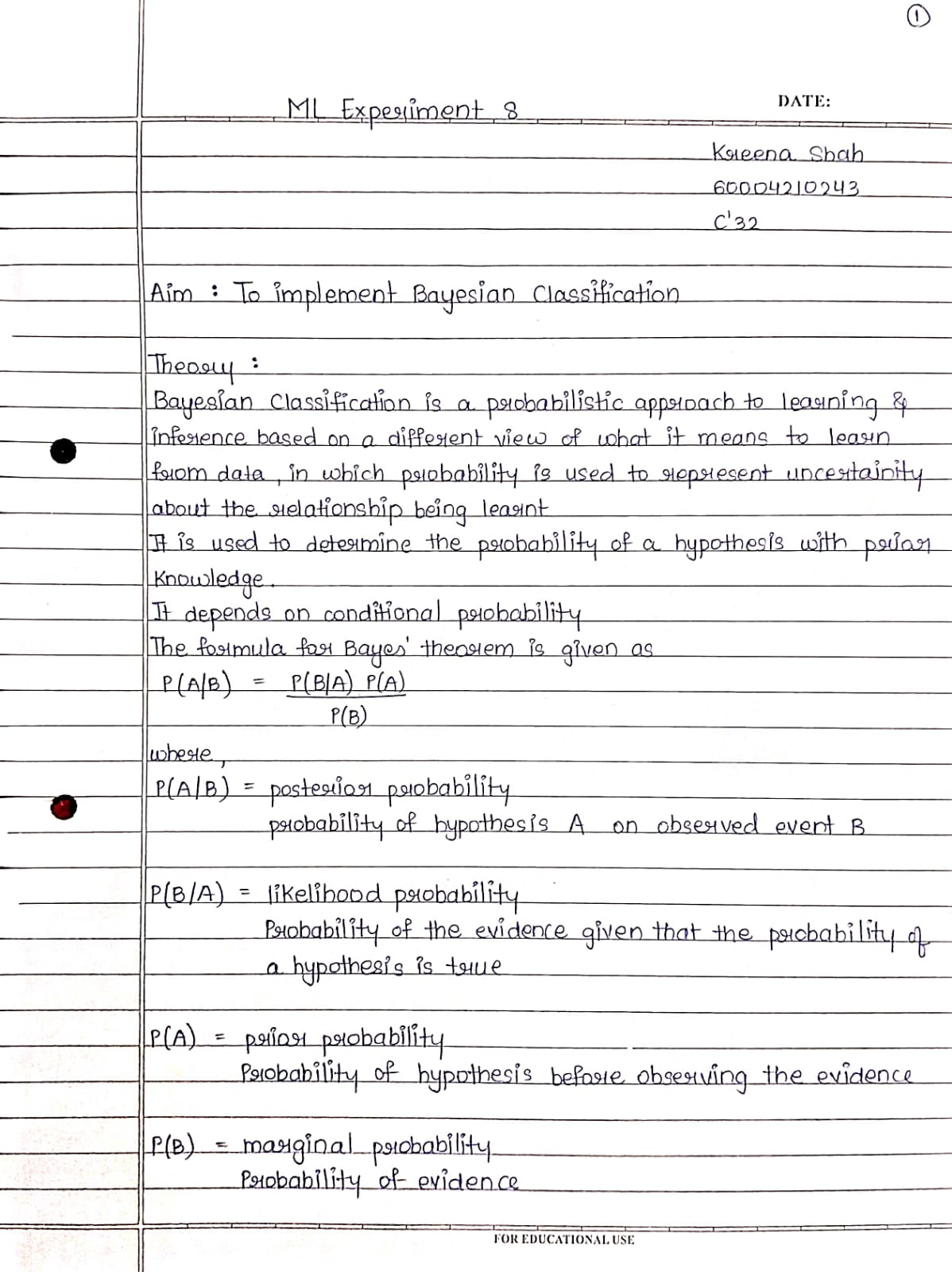
print(accuracy)

from sklearn.metrics import confusion\_matrix

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

###### 

###### ML Experiment 8



from sklearn.datasets import load\_breast\_cancer from sklearn.model\_selection import train\_test\_split from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score data = load\_breast\_cancer()

X = data.data y = data.target X

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

classifier = SVC(kernel=' ') classifier.fit(X\_train, Y\_train) Y\_pred = classifier.predict(X\_test)

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

print("Accuracy:", accuracy)

from sklearn.metrics import confusion\_matrix

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