Practical 1(object oriented concepts)

class Account:

amount = 0

def \_\_init\_\_(self):

self.balance = 1000.0

print("WELCOME")

self.Name = input("Enter the name of Account Holder: ")

self.Account\_no = input("Enter the Account no.")

def deposit(self):

amount = float(input("\nEnter Amount to be Deposited: "))

self.balance += amount

print("\nYour amount is deposited successfully")

print("\nTotal amount is:", self.balance)

self.ask\_for\_another\_task()

def withdraw(self):

amount = float(input("\nEnter Amount to be Withdrawn: "))

if self.balance >= amount:

self.balance -= amount

print("\nWithdrawal amount is:", amount)

print("Available balance:", self.balance)

else:

print("You cannot withdraw due to insufficient balance:", self.balance)

self.ask\_for\_another\_task()

def display(self):

print("\nNet Available Balance =", self.balance)

print("Name of Account holder:", self.Name, "\nAccount no.:", self.Account\_no)

self.ask\_for\_another\_task()

def ask\_for\_another\_task(self):

answer = input("\nDo you want to perform another task? (yes/no): ")

if answer.lower() == "yes":

self.perform\_task()

else:

print("Exiting program.")

def perform\_task(self):

choice = input("1 for DEPOSIT\n2 for Withdrawal\n3 to check Balance\n")

if choice == '1':

self.deposit()

elif choice == '2':

self.withdraw()

elif choice == '3':

self.display()

else:

print("Invalid choice. Please enter a valid option.")

self.perform\_task()

s = Account()

s.perform\_task()

Practical 2(Numpy)

#replace numpy array

import numpy as np

a = np.array([34, 56, 32, 24, 58, 98])

a[a >= 50] = 15.50

print("One-D array elements replacement:", a)

a = np.array([[34, 56, 32, 24, 58, 98], [34, 56, 32, 24, 58, 98]])

a[a >= 50] = 15.50

print("\nTwo-D array elements replacement:\n", a)

a = np.array([[[1, 2, 49, 4, 5], [6, 7, 50, 9, 10]], [[11, 12, 89, 14, 15], [16, 17, 78, 10, 20]]])

a[a >= 50] = 15.50

print("\nThree-D array elements replacement:\n", a)

# non numeric values

n\_arr = np.array([[10.5, 22.5, 3.8], [41, np.nan, np.nan]])

print("Given array:")

print(n\_arr)

# Remove rows containing non-numeric values

n\_arr = n\_arr[~np.isnan(n\_arr).any(axis=1)]

print("\nArray after removing rows with non-numeric values:")

print(n\_arr)

Practical-3(pandas)

import pandas as pd

df = pd.read\_csv("C:\\Users\\student\\Desktop\\book.csv")

print(df)

print(df.to\_string())

print(df.head(4))

print(df.tail(1))

print(df.info())

Practical-4

import csv

with open('student.csv', 'r') as csvfile:

csvreader = csv.reader(csvfile)

for row in csvreader:

print(row)

import csv

with open('student.csv','w',newline='')as csvfile:

fieldnames=['Roll no','Name','Total marks']

writer=csv.DictWriter(csvfile,fieldnames=fieldnames)

writer.writeheader()

writer.writerow({'Roll no':'1','Name':'shreyash','Total marks':'100'})

writer.writerow({'Roll no':'2','Name':'shatayu','Total marks':'80'})

writer.writerow({'Roll no':'3','Name':'sohel','Total marks':'45'})

writer.writerow({'Roll no':'4','Name':'rajui','Total marks':'78'})

writer.writerow({'Roll no':'5','Name':'joijo','Total marks':'46'})

writer.writerow({'Roll no':'6','Name':'fdssad','Total marks':'56'})

writer.writerow({'Roll no':'7','Name':'safdws','Total marks':'45'})

import csv

with open('student.csv','r')as csvfile:

shreyash=csv.reader(csvfile)

count=0

avg=0

mini=54

maxi=0

for row in shreyash:

if count!=0:

if mini>int(row[2]):

mini=int(row[2])

if maxi<int(row[2]):

maxi=int(row[2])

avg=avg+int(row[2])

count=1

print('Average=',avg/6)

print('Min=',mini)

print('Max=',maxi)

import csv

with open('student.csv','r')as csvfile:

shreyash=csv.reader(csvfile)

count=0

for row in shreyash:

if count!=0:

if int(row[2]>75):

print(row)

count=1

Practical-5(Matplotlib)

import matplotlib.pyplot as plt

# Bar chart

fig, ax = plt.subplots()

name = ['anushka', 'sakshi', 'kunika']

marks = [10, 15, 12]

c = ['red', 'yellow', 'black']

ax.bar(name, marks, color=c)

ax.set\_xlabel('Name')

ax.set\_ylabel('Marks')

ax.set\_title('Marks Scored by Students')

plt.show()

# Bar chart with data from CSV file

import matplotlib.pyplot as plt

import pandas as pd

# Read data from CSV file

data = pd.read\_csv('C:\\Users\\student\\Downloads\\sample.csv')

df = pd.DataFrame(data)

X = list(df.iloc[:, 0])

Y = list(df.iloc[:, 1])

# Plot the data using bar() method

i = ['red', 'yellow', 'orange', 'blue']

plt.bar(X, Y, color=i)

plt.title("Distribution of Salary")

plt.xlabel("Name")

plt.ylabel("Salary")

# Show the plot

plt.show()

# Pie chart

import matplotlib.pyplot as plt

import numpy as np

y = np.array([20, 30, 40, 35, 8, 20])

mylabels = ["sakshi", "kunika", "anushka", "manthan", "Pandu", "hemlo"]

plt.pie(y, labels=mylabels)

plt.title("Distribution of Something")

plt.show()

Practical 6(scipy)

import numpy as np

from scipy.sparse import csr\_matrix

from scipy.optimize import root

# Create a 1D sparse array

sparse\_array = csr\_matrix([0, 0, 1, 0, 2, 0, 0, 3, 0])

non\_zero\_positions = sparse\_array.nonzero()[1]

print("Positions of non-zero elements:", non\_zero\_positions)

non\_zero\_count = len(non\_zero\_positions)

print("Number of non-zero elements:", non\_zero\_count)

# Create a 2D sparse array

sparse\_array = csr\_matrix([[0, 0, 1],

[0, 2, 0],

[0, 0, 3]])

non\_zero\_positions = sparse\_array.nonzero()

print("Positions of non-zero elements:")

for row, col in zip(non\_zero\_positions[0], non\_zero\_positions[1]):

print(f"({row}, {col})")

non\_zero\_count = len(non\_zero\_positions[0])

print("Number of non-zero elements:", non\_zero\_count)

# Define the polynomial equation

def polynomial\_equation(x):

return x\*\*3 - 2\*x\*\*2 - 5\*x + 6

# Find the roots of the polynomial equation

solution = root(polynomial\_equation, x0=[-10, -5, 5])

# Display the roots

roots = solution.x

print("Roots of the polynomial equation:", roots)

Practical-7(scikit)

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

dataset = pd.read\_csv('/content/suv\_data - suv\_data.csv')

print(dataset)

# input

x = dataset.iloc[:, [2, 3]].values

# output

y = dataset.iloc[:, 4].values

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

xtrain, xtest, ytrain, ytest = train\_test\_split(

x, y, test\_size=0.25, random\_state=0)

from sklearn.preprocessing import StandardScaler

sc\_x = StandardScaler()

xtrain = sc\_x.fit\_transform(xtrain)

xtest = sc\_x.transform(xtest)

print (xtrain[0:10, :])

from sklearn.linear\_model import LogisticRegression

classifier = LogisticRegression(random\_state = 0)

classifier.fit(xtrain, ytrain)

y\_pred = classifier.predict(xtest)

from sklearn.metrics import confusion\_matrix

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

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

from sklearn.metrics import accuracy\_score

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

from matplotlib.colors import ListedColormap

X\_set, y\_set = xtest, ytest

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1,

stop = X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1,

stop = X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(

np.array([X1.ravel(), X2.ravel()]).T).reshape(

X1.shape), alpha = 0.75, cmap = ListedColormap(('blue', 'orange')))

plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)):

plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('yellow', 'black'))(i), label = j)

plt.title('Classifier (Test set)')

plt.xlabel('Age')

plt.ylabel('Estimated Salary')

plt.legend()

plt.show()

Practical-8(tensorflow)

pip install tensorflow

import pandas as pd

import numpy as np

import os

# Plotting libraries

import matplotlib.pyplot as plt

# SKLearn libraries

from sklearn.preprocessing import LabelEncoder

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

# Tensorflow libraries

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.models import Sequential

# Data file path

FILE\_PATH = '/Users/SAKSHI/Downloads/IRIS.csv'

# Dataframe from csv file

iris\_data = pd.read\_csv(FILE\_PATH, header=0)

iris\_data.info()

print("=="\*40)

iris\_data.head(10)

X = iris\_data.loc[:, iris\_data.columns != 'species']

y = iris\_data.loc[:, ['species']]

y\_enc = LabelEncoder().fit\_transform(y)

# Converting the label into a matrix form

y\_label = tf.keras.utils.to\_categorical(y\_enc)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y\_label, test\_size=0.3)

print(f"Train shape : {X\_train.shape}, Y Train : {y\_train.shape}")

print(X\_train.shape[1:])

def get\_model():

model = Sequential([

keras.layers.Input(shape=X\_train.shape[1:]),

keras.layers.Dense(1000, activation='relu'),

keras.layers.Dense(500, activation='relu',),

keras.layers.Dense(300, activation='relu'),

keras.layers.Dropout(0.2),

keras.layers.Dense(3, activation='softmax')

])

return model

model = get\_model()

# Compile the model

model.compile(optimizer='adam',

loss=keras.losses.CategoricalCrossentropy(),

metrics=['accuracy'])

model.summary()

history = model.fit(X\_train, y\_train, epochs=30, validation\_data=(X\_test, y\_test), verbose=1)

model.evaluate(X\_test, y\_test)

pd.DataFrame(history.history).plot(figsize=(10,6))

plt.grid(True)

plt.gca().set\_ylim(0, 1)

plt.show()

new\_data, y\_actual = X\_test[:3], y\_test[:3]

y\_proba = model.predict(new\_data)

print(f"Actual data : {y\_actual}")

for pred in y\_proba:

print(np.argmax(pred))