**Machine Intelligence Lab**

**Week-1**

**Name:Thushar M**

**SRN:PES1UG20CS471**

**Section:H**

● create\_numpy\_ones\_array(shape)

■Input : tuple (x,y)

■ Output: numpy array of the shape (x,y) with 1 at all position

**Code:**

from fileinput import filename

import numpy as np

import pandas as pd

*#input: tuple (x,y)    x,y:int*

def create\_numpy\_ones\_array(shape):

*#return a numpy array with one at all index*

    array=np.ones(shape)

*#TODO*

    return array

● create\_numpy\_zeros\_array(shape)

■ Input : tuple (x,y)

■ Output: numpy array of the shape (x,y) with 0 at all position

**Code:**

def create\_numpy\_zeros\_array(shape):

*#return a numpy array with zeroes at all index*

    array=np.zeros(shape)

*#TODO*

    return array

● create\_identity\_numpy\_array(order)

■ Input : int

■ Output: Identity matrix in the form of numpy array of dimension order x order

**Code:**

def create\_identity\_numpy\_array(order):

*#return a identity numpy array of the defined order*

    array=None

    array=np.identity(order)

    return array

● matrix\_cofactor(array)

■ Input: numpy array

■ Output: cofactor matrix of the input in the form of numpy array

**Code:**

def matrix\_cofactor(array):

*#return cofactor matrix of the given array*

    det=np.linalg.det(array)

    if(det!=0):

        cofactor=None

        cofactor=np.linalg.inv(array).T\*det

*#TODO*

        return cofactor

    else:

        return None

● f1(X1,coef1,X2,coef2,seed1,seed2,seed3,shape1,shape2)

■ Input: (numpy array, int ,numpy array, int , int , int , int ,tuple,tuple)

■ Perform W1 x (X1 \*\* coef1) + W2 x (X2 \*\* coef2) +b

■ where W1 is random matrix of shape shape1 with seed1

■ where W2 is random matrix of shape shape2 with seed2

■ if dimension mismatch occur return -1

■ Output: computed function(numpy array) or -1

**Code:**

def f1(X1,coef1,X2,coef2,seed1,seed2,seed3,shape1,shape2):

*#note: shape is of the forest (x1,x2)*

*#return W1 x (X1 \*\* coef1) + W2 x (X2 \*\* coef2) +b*

*# where W1 is random matrix of shape shape1 with seed1*

    np.random.seed(seed1)

    W1=np.random.rand(\*shape1)

*# where W2 is random matrix of shape shape2 with seed2*

    np.random.seed(seed2)

    W2=np.random.rand(\*shape2)

    if(np.shape(W1)!=np.shape(W2)):

        return -1

    ans=np.dot(W1,np.linalg.matrix\_power(X1,coef1))+np.dot(W2,np.linalg.matrix\_power(X2,coef2))

*# where B is a random matrix of comaptible shape with seed3*

    shape\_ans=np.shape(ans)

    np.random.seed(seed3)

    B=np.random.rand(\*shape\_ans)

*# if dimension mismatch occur return -1*

    ans+=B

    return ans

● fill\_with\_mode(filename, column)

■ Input: (str, str)

■ Fill the missing values(NaN) in a column with the mode of that column

■ output: df: Pandas DataFrame object.(Representing entire data and where 'column' does not contain NaNvalues)

**Code:**

def fill\_with\_mode(filename, column):

    """

    Fill the missing values(NaN) in a column with the mode of that column

    Args:

        filename: Name of the CSV file.

        column: Name of the column to fill

    Returns:

        df: Pandas DataFrame object.

        (Representing entire data and where 'column' does not contain NaN values)

        (Filled with above mentioned rules)

    """

    df=pd.read\_csv(filename)

    df[column].fillna(df[column].mode()[0],inplace=True)

    return df

● fill\_with\_group\_average(df, group, column)

■ Input: (DataFrame,str, str)

■ Fill the missing values(NaN) in ‘column’ with the mean value of the group the row belongs to.

■ output: df: Pandas DataFrame object.(Representing entire data and where 'column' does not contain NaNvalues)

**Code:**

def fill\_with\_group\_average(df, group, column):

    """

    Fill the missing values(NaN) in column with the mean value of the

    group the row belongs to.

    The rows are grouped based on the values of another column

    Args:

        df: A pandas DataFrame object representing the data.

        group: The column to group the rows with

        column: Name of the column to fill

    Returns:

        df: Pandas DataFrame object.

        (Representing entire data and where 'column' does not contain NaN values)

        (Filled with above mentioned rules)

    """

    df[column].fillna(df.groupby(group)[column].transform(lambda x:x.fillna(x.mean())),inplace=True)

    return df

● get\_rows\_greater\_than\_avg(df, column)

■ Input: (DataFrame, str)

■ Return all the rows(with all columns) where the value in a certain 'column' is greater than the average value of that column.

■ output: df: Pandas DataFrame object.

**Code:**

def get\_rows\_greater\_than\_avg(df, column):

    """

    Return all the rows(with all columns) where the value in a certain 'column'

    is greater than the average value of that column.

    row where row.column > mean(data.column)

    Args:

        df: A pandas DataFrame object representing the data.

        column: Name of the column to fill

    Returns:

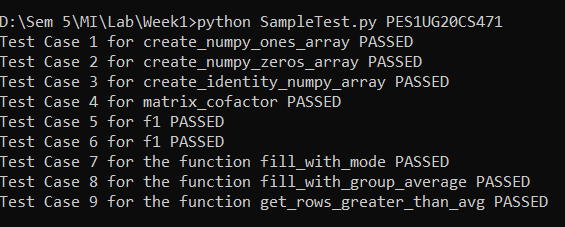
        df: Pandas DataFrame object.

    """

    df=df[df[column]>df[column].mean()]

    return df

**Output:**

****