2a

import numpy as np

# Defining 1D array

my1DArray = np.array([1, 8, 27, 64]) # Corrected "l" to "1" and added closing bracket

print("1D Array:")

print(my1DArray)

# Defining and printing 2D array

my2DArray = np.array([[1, 2, 3, 4], [2, 4, 9, 16], [4, 8, 18, 32]]) # Corrected "l" to "1"

print("\n2D Array:")

print(my2DArray)

# Defining and printing 3D array

my3DArray = np.array([[[1, 2, 3, 4], [5, 6, 7, 8]], [[1, 2, 3, 4], [9, 10, 11, 12]]])

print("\n3D Array:")

print(my3DArray)

# Print out memory address

print("\nMemory address of my2DArray:")

print(my2DArray.data)

# Print the shape of the array

print("\nShape of my2DArray:")

print(my2DArray.shape)

# Print out the data type of the array

print("\nData type of my2DArray:")

print(my2DArray.dtype)

# Print the stride of the array

print("\nStride of my2DArray:")

print(my2DArray.strides)

# (ii) Creation of an array using built-in NumPy functions

# Array of ones

ones = np.ones((3, 4))

print("\nArray of ones:")

print(ones)

# Array of zeros

zeros = np.zeros((2, 3, 4), dtype=np.int16)

print("\nArray of zeros:")

print(zeros)

# Array with random values

random\_values = np.random.random((2, 2))

print("\nArray with random values:")

print(random\_values)

# Empty array

emptyArray = np.empty((3, 2))

print("\nEmpty array:")

print(emptyArray)

# Full array

fullArray = np.full((2, 2), 7)

print("\nFull array:")

print(fullArray)

# Array of evenly spaced values

evenSpacedArray = np.arange(10, 25, 5)

print("\nArray of evenly spaced values (using arange):")

print(evenSpacedArray)

# Array of evenly spaced values

evenSpacedArray2 = np.linspace(0, 2, 9)

print("\nArray of evenly spaced values (using linspace):")

print(evenSpacedArray2)

# (iii) Performing file operations with NumPy arrays

# Initialize an array

arr = np.array([[[11, 11, 9, 9], [11, 0, 2, 0]], [[10, 14, 9, 14], [0, 1, 11, 11]]])

# Open a binary file in write mode and save the array

with open("arr.npy", "wb") as file: # Fixed file extension and mode

np.save(file, arr)

# Open the file in read binary mode and read the array

with open("arr.npy", "rb") as file:

arr1 = np.load(file)

print("\nLoaded array from file:")

print(arr1)

o/p

1D Array:

[ 1 8 27 64]

2D Array:

[[ 1 2 3 4]

[ 2 4 9 16]

[ 4 8 18 32]]

3D Array:

[[[ 1 2 3 4]

[ 5 6 7 8]]

[[ 1 2 3 4]

[ 9 10 11 12]]]

Memory address of my2DArray:

<memory at 0x7fxxxxxxx>

Shape of my2DArray:

(3, 4)

Data type of my2DArray:

int64

Stride of my2DArray:

(32, 8)

Array of ones:

[[1. 1. 1. 1.]

[1. 1. 1. 1.]

[1. 1. 1. 1.]]

Array of zeros:

[[[0 0 0 0]

[0 0 0 0]

[0 0 0 0]]

[[0 0 0 0]

[0 0 0 0]

[0 0 0 0]]]

Array with random values:

[[0.12345678 0.87654321]

[0.56789012 0.34567890]] # These values are random and will vary each run

Empty array:

[[1.345e-123 4.567e-234]

[2.345e+001 1.234e+002]

[3.456e+123 9.876e+098]] # Values depend on uninitialized memory

Full array:

[[7 7]

[7 7]]

Array of evenly spaced values (using arange):

[10 15 20]

Array of evenly spaced values (using linspace):

[0. 0.25 0.5 0.75 1. 1.25 1.5 1.75 2. ]

Loaded array from file:

[[[11 11 9 9]

[11 0 2 0]]

[[10 14 9 14]

[ 0 1 11 11]]]

2b

import numpy as np

# Define the first array

a = np.arange(9, dtype=np.float\_).reshape(3, 3)

print('First array:')

print(a)

print('\n')

# Define the second array

b = np.array([10, 10, 10]) # Corrected "l0" to "10"

print('Second array:')

print(b)

print('\n')

# Perform element-wise addition

print('Add the two arrays:')

print(np.add(a, b))

print('\n')

# Perform element-wise subtraction

print('Subtract the two arrays:')

print(np.subtract(a, b))

print('\n')

# Perform element-wise multiplication

print('Multiply the two arrays:')

print(np.multiply(a, b))

print('\n')

# Perform element-wise division

print('Divide the two arrays:')

print(np.divide(a, b))

o/p

First array:

[[0. 1. 2.]

[3. 4. 5.]

[6. 7. 8.]]

Second array:

[10 10 10]

Add the two arrays:

[[10. 11. 12.]

[13. 14. 15.]

[16. 17. 18.]]

Subtract the two arrays:

[[-10. -9. -8.]

[ -7. -6. -5.]

[ -4. -3. -2.]]

Multiply the two arrays:

[[ 0. 10. 20.]

[30. 40. 50.]

[60. 70. 80.]]

Divide the two arrays:

[[0. 0.1 0.2]

[0.3 0.4 0.5]

[0.6 0.7 0.8]]

3

import numpy as np

import pandas as pd

# Display Pandas version

print("Pandas Version:", pd.\_\_version\_\_)

# Configure display options

pd.set\_option('display.max\_columns', 500)

pd.set\_option('display.max\_rows', 500)

# Create a Pandas Series

series = pd.Series([2, 3, 7, 11, 13, 17, 19, 23])

print("\nSeries:")

print(series)

# Create a DataFrame

series\_df = pd.DataFrame({

'A': range(1, 5), # Corrected "l" to "1"

'B': pd.Timestamp('20190526'),

'C': pd.Series(5, index=list(range(4)), dtype='float64'),

'D': np.array([3] \* 4, dtype='int64'),

'E': pd.Categorical(["Depression", "Social Anxiety", "Bipolar Disorder", "Eating Disorder"]),

'F': 'Mental health',

'G': 'is challenging'

})

print("\nDataFrame from Series:")

print(series\_df)

# Create a DataFrame from a list of dictionaries

dict\_df = [{'A': 'Apple', 'B': 'Ball'}, {'A': 'Aeroplane', 'B': 'Bat', 'C': 'Cat'}]

dict\_df = pd.DataFrame(dict\_df) # Fixed variable name "diet\_df" to "dict\_df"

print("\nDataFrame from Dictionary:")

print(dict\_df)

# Create a DataFrame from an N-Dimensional structure

sdf = {

'County': ['Ostfold', 'Hordaland', 'Oslo', 'Hedmark', 'Oppland', 'Buskerud'],

'ISO-Code': [1, 2, 3, 4, 5, 6],

'Area': [4180.69, 4917.94, 454.07, 27397.76, 25192.10, 14910.94],

'Administrative centre': ["Sarpsborg", "Bergen", "Oslo", "Hamar", "Lillehammer", "Drammen"]

}

sdf = pd.DataFrame(sdf)

print("\nDataFrame from N-Dimensional Arrays:")

print(sdf)

# Define column names

columns = [

'age', 'workclass', 'fnlwgt', 'education', 'education\_num', 'marital\_status',

'occupation', 'relationship', 'ethnicity', 'gender', 'capital\_gain',

'capital\_loss', 'hours\_per\_week', 'country\_of\_origin', 'income'

]

# Load dataset into a DataFrame

df = pd.read\_csv('http://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data', names=columns)

# Display the first 10 rows

print("\nFirst 10 rows of the loaded dataset:")

print(df.head(10))

4a

import pandas as pd

# Reading CSV file

df = pd.read\_csv('example.csv')

print("CSV Input:")

print(df)

o/p

a b c d

0 0 1 2 3

1 4 5 6 7

2 8 9 10 11

3 12 13 14 15

# Writing to CSV file without index

df.to\_csv('example.csv', index=False)

# Reading an Excel file

df\_excel = pd.read\_excel('Excel\_Sample.xlsx', sheet\_name='Sheet 1')

print("Excel Input:")

print(df\_excel)

o/p

a b c d

0 0 1 2 3

1 4 5 6 7

2 8 9 10 11

3 12 13 14 15

# Writing to Excel file

df.to\_excel('Excel\_Sample.xlsx', sheet\_name='Sheet 1', index=False)

pip install lxml html5lib beautifulsoup4

# Reading tables from an HTML page

url = "https://www.fdic.gov/resources/resolutions/bank-failures/failed-bank-list"

df\_list = pd.read\_html(url) # Reads all tables from the webpage

# Accessing the first table

df\_html = df\_list[0]

print("HTML Input - First Table:")

print(df\_html.head()) # Displays the first few rows of the table

Bank Name City ST CERT \

0 First Cornerstone Bank King of Prussia PA 35312

1 Trust Company Bank Memphis TN 9956

2 North Milwaukee State Bank Milwaukee WI 20364

...

Closing Date Updated Date ... <other columns>

4b

import pandas as pd

from pandas import DataFrame

from sklearn.datasets import load\_iris # Corrected the typo from "skleam" to "sklearn"

# Load the iris dataset

iris\_obj = load\_iris()

# Preview the dataset (features and target combined into a single DataFrame)

iris = (

DataFrame(iris\_obj.data, columns=iris\_obj.feature\_names) # Feature data

.join(DataFrame(iris\_obj.target, columns=["species"])) # Target data

)

# Print the Iris dataset

print("Iris DataFrame:\n", iris)

# Feature names

print("\nFeature Names:")

print(iris\_obj.feature\_names)

# Various descriptive statistics

print("\nRow Count:")

print(iris.count())

print("\nMean:")

print(iris.mean(numeric\_only=True))

print("\nMedian:")

print(iris.median(numeric\_only=True))

print("\nVariance:")

print(iris.var(numeric\_only=True))

print("\nStandard Deviation:")

print(iris.std(numeric\_only=True))

print("\nMaximum Values:")

print(iris.max(numeric\_only=True))

print("\nMinimum Values:")

print(iris.min(numeric\_only=True))

print("\nSummary Statistics:")

print(iris.describe())

o/p

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) species

0 5.1 3.5 1.4 0.2 0

1 4.9 3.0 1.4 0.2 0

2 4.7 3.2 1.3 0.2 0

3 4.6 3.1 1.5 0.2 0

4 5.0 3.6 1.4 0.2 0

['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']

sepal length (cm) 150

sepal width (cm) 150

petal length (cm) 150

petal width (cm) 150

species 150

dtype: int64

sepal length (cm) 5.843333

sepal width (cm) 3.057333

petal length (cm) 3.758000

petal width (cm) 1.199333

species 1.000000

dtype: float64

sepal length (cm) 5.8

sepal width (cm) 3.0

petal length (cm) 4.35

petal width (cm) 1.3

species 1.0

dtype: float64

sepal length (cm) 0.685694

sepal width (cm) 0.189979

petal length (cm) 3.116278

petal width (cm) 0.581006

species 0.671141

dtype: float64

sepal length (cm) 0.828066

sepal width (cm) 0.435866

petal length (cm) 1.765298

petal width (cm) 0.762238

species 0.819232

dtype: float64

sepal length (cm) 7.9

sepal width (cm) 4.4

petal length (cm) 6.9

petal width (cm) 2.5

species 2.0

dtype: float64

sepal length (cm) 4.3

sepal width (cm) 2.0

petal length (cm) 1.0

petal width (cm) 0.1

species 0.0

dtype: float64

sepal length (cm) sepal width (cm) petal length (cm) petal width (cm) species

count 150.000000 150.000000 150.000000 150.000000 150.000000

mean 5.843333 3.057333 3.758000 1.199333 1.000000

std 0.828066 0.435866 1.765298 0.762238 0.819232

min 4.300000 2.000000 1.000000 0.100000 0.000000

25% 5.100000 2.800000 1.600000 0.300000 0.000000

50% 5.800000 3.000000 4.350000 1.300000 1.000000

75% 6.400000 3.300000 5.100000 1.800000 2.000000

max 7.900000 4.400000 6.900000 2.500000 2.000000

5a

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

sns.set\_style('darkgrid')

%matplotlib inline # Ensure inline plotting for Jupyter Notebook

from matplotlib.ticker import FormatStrFormatter

import warnings

warnings.filterwarnings('ignore')

# Load dataset

df = pd.read\_csv('C:/Users/kirub/Documents/Learning/Untitled Folder/diabetes.csv')

# Display first few rows

print("First five rows of the dataset:")

print(df.head())

# Dataset dimensions

print("\nShape of the dataset:")

print(df.shape)

# Data types

print("\nData types:")

print(df.dtypes)

# Convert Outcome column to boolean

df['Outcome'] = df['Outcome'].astype('bool')

print("\nUpdated Data Type for 'Outcome':")

print(df.dtypes['Outcome'])

# Dataset info and descriptive statistics

print("\nDataset Info:")

df.info()

print("\nDescriptive Statistics (Transposed):")

print(df.describe().T)

# Outcome variable frequency count

print("\nFrequency Count of 'Outcome':")

outcome\_counts = df['Outcome'].value\_counts()

print(outcome\_counts)

# Measures of central tendency and dispersion

print("\nMean:\n", df.mean(numeric\_only=True))

print("\nMedian:\n", df.median(numeric\_only=True))

print("\nMode:\n", df.mode().iloc[0])

print("\nVariance:\n", df.var(numeric\_only=True))

print("\nStandard Deviation:\n", df.std(numeric\_only=True))

# Kurtosis

print("\nKurtosis (Overall):\n", df.kurtosis(axis=0, skipna=True))

print("\nKurtosis of 'Outcome':\n", df['Outcome'].kurtosis(axis=0, skipna=True))

# Skewness

print("\nSkewness (Overall):\n", df.skew(axis=0, skipna=True))

print("\nSkewness for Rows (first 5 rows):\n", df.skew(axis=1, skipna=True).head())

# Analysis of 'Pregnancies' variable

preg\_proportion = np.array(df['Pregnancies'].value\_counts())

preg\_month = np.array(df['Pregnancies'].value\_counts().index)

preg\_proportion\_perc = np.array(np.round(preg\_proportion / sum(preg\_proportion), 3) \* 100, dtype=int)

preg = pd.DataFrame({

'month': preg\_month,

'count\_of\_preg\_prop': preg\_proportion,

'percentage\_proportion': preg\_proportion\_perc

})

preg.set\_index(['month'], inplace=True)

print("\nPregnancy Proportions:")

print(preg.head(10))

# Visualization

plt.figure(figsize=(12, 5))

# Countplot for 'Outcome'

plt.subplot(1, 3, 1)

sns.countplot(x=df['Outcome'])

plt.title('Outcome Count')

# Distribution plot for 'Pregnancies'

plt.subplot(1, 3, 2)

sns.histplot(df['Pregnancies'], kde=True, bins=10)

plt.title('Distribution of Pregnancies')

# Boxplot for 'Pregnancies'

plt.subplot(1, 3, 3)

sns.boxplot(x=df['Pregnancies'])

plt.title('Boxplot of Pregnancies')

plt.tight\_layout()

plt.show()

5b

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

# Set style for seaborn

sns.set\_style('darkgrid')

# Ignore warnings

import warnings

warnings.filterwarnings('ignore')

# Load dataset

df = pd.read\_csv('C:/Users/diabetes.csv')

# Preview data

print("First five rows of the dataset:")

print(df.head())

# Dataset dimensions and data types

print("\nShape of the dataset:", df.shape)

print("\nData types:\n", df.dtypes)

# Convert Outcome column to boolean

df['Outcome'] = df['Outcome'].astype('bool')

# Create subplots for visualizations

fig, axes = plt.subplots(nrows=3, ncols=2, dpi=120, figsize=(8, 6))

# Countplot for Pregnancies

sns.countplot(x='Pregnancies', data=df, ax=axes[0][0], color='green')

axes[0][0].set\_title('Count', fontdict={'fontsize': 8})

axes[0][0].set\_xlabel('Number of Pregnancies', fontdict={'fontsize': 7})

axes[0][0].set\_ylabel('Count', fontdict={'fontsize': 7})

# Countplot with hue for Outcome

sns.countplot(x='Pregnancies', data=df, hue='Outcome', ax=axes[0][1])

axes[0][1].set\_title('Diabetes vs Non-Diabetes', fontdict={'fontsize': 8})

axes[0][1].set\_xlabel('Number of Pregnancies', fontdict={'fontsize': 7})

axes[0][1].set\_ylabel('Count', fontdict={'fontsize': 7})

axes[0][1].legend(loc=1, fontsize=6)

# Distribution plot for Pregnancies

sns.histplot(df['Pregnancies'], kde=True, ax=axes[1][0])

axes[1][0].set\_title('Pregnancies Distribution', fontdict={'fontsize': 8})

axes[1][0].set\_xlabel('Number of Pregnancies', fontdict={'fontsize': 7})

axes[1][0].set\_ylabel('Frequency/Distribution', fontdict={'fontsize': 7})

# Histogram for Diabetes vs Non-Diabetes

df[df['Outcome'] == False]['Pregnancies'].plot.hist(ax=axes[1][1], label='Non-Diabetic', alpha=0.6)

df[df['Outcome'] == True]['Pregnancies'].plot.hist(ax=axes[1][1], label='Diabetic', alpha=0.6)

axes[1][1].set\_title('Diabetes vs Non-Diabetes', fontdict={'fontsize': 8})

axes[1][1].set\_xlabel('Number of Pregnancies', fontdict={'fontsize': 7})

axes[1][1].set\_ylabel('Frequency/Distribution', fontdict={'fontsize': 7})

axes[1][1].legend(loc=1, fontsize=6)

# Boxplot for Pregnancies

sns.boxplot(y=df['Pregnancies'], ax=axes[2][0], orient='v')

axes[2][0].set\_title('Pregnancies Boxplot', fontdict={'fontsize': 8})

axes[2][0].set\_xlabel('Pregnancies', fontdict={'fontsize': 7})

axes[2][0].set\_ylabel('Five Point Summary', fontdict={'fontsize': 7})

# Boxplot for Pregnancies by Outcome

sns.boxplot(x='Outcome', y='Pregnancies', data=df, ax=axes[2][1])

axes[2][1].set\_title('Diabetes vs Non-Diabetes (Boxplot)', fontdict={'fontsize': 8})

axes[2][1].set\_xlabel('Outcome', fontdict={'fontsize': 7})

axes[2][1].set\_ylabel('Number of Pregnancies', fontdict={'fontsize': 7})

axes[2][1].set\_xticks([0, 1])

axes[2][1].set\_xticklabels(['Non-Diabetic', 'Diabetic'], fontsize=7)

# Adjust layout

plt.tight\_layout()

# Show the plots

plt.show()

6

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# Load the Dataset

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"

column\_names = ['sepal\_length', 'sepal\_width', 'petal\_length', 'petal\_width', 'species']

data = pd.read\_csv(url, header=None, names=column\_names)

# Normal Curves

plt.figure(figsize=(10, 6))

sns.kdeplot(data['sepal\_length'], color='blue', label='Sepal Length', fill=True)

sns.kdeplot(data['sepal\_width'], color='orange', label='Sepal Width', fill=True)

plt.title('Normal Curves for Sepal Dimensions')

plt.xlabel('Length/Width')

plt.ylabel('Density')

plt.legend()

plt.show()

# Density and Contour Plots

plt.figure(figsize=(10, 6))

sns.kdeplot(x=data['sepal\_length'], y=data['sepal\_width'], cmap='Blues', fill=True)

plt.title('Density and Contour Plot')

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.show()

# Correlation and Scatter Plots

plt.figure(figsize=(10, 6))

sns.scatterplot(x='sepal\_length', y='sepal\_width', hue='species', data=data)

plt.title('Scatter Plot of Sepal Dimensions')

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.show()

# Histograms

plt.figure(figsize=(10, 6))

data['sepal\_length'].hist(bins=20, color='lightblue', edgecolor='black')

plt.title('Histogram of Sepal Length')

plt.xlabel('Sepal Length')

plt.ylabel('Frequency')

plt.show()

# 3D Scatter Plot

fig = plt.figure(figsize=(10, 6))

ax = fig.add\_subplot(111, projection='3d')

ax.scatter(data['sepal\_length'], data['sepal\_width'], data['petal\_length'],

c=data['species'].factorize()[0], cmap='viridis', marker='o')

ax.set\_xlabel('Sepal Length')

ax.set\_ylabel('Sepal Width')

ax.set\_zlabel('Petal Length')

plt.title('3D Scatter Plot')

plt.show()

# Boxplots

plt.figure(figsize=(10, 6))

sns.boxplot(x='species', y='sepal\_length', data=data)

plt.title('Boxplot of Sepal Length by Species')

plt.show()

# Pairplot

sns.pairplot(data, hue='species')

plt.show()

7

import matplotlib.pyplot as plt

from mpl\_toolkits.basemap import Basemap

# Step 2: Load Geographic Data

# Example city data: list of cities with latitude and longitude

cities = {

'New York': (40.7128, -74.0060),

'Los Angeles': (34.0522, -118.2437),

'Chicago': (41.8781, -87.6298),

'Houston': (29.7604, -95.3698),

'Phoenix': (33.4484, -112.0740)

}

# Step 3: Set Up the Base Map

plt.figure(figsize=(10, 8))

map = Basemap(projection='merc',

llcrnrlat=20, urcrnrlat=50,

llcrnrlon=-130, urcrnrlon=-60,

lat\_ts=20, resolution='i')

# Draw map details

map.drawcoastlines()

map.drawcountries()

map.drawstates()

map.drawmapboundary(fill\_color='lightblue')

map.fillcontinents(color='lightgreen', lake\_color='lightblue')

# Step 4: Plot Data Points

for city, (lat, lon) in cities.items():

x, y = map(lon, lat) # Convert lat/lon to x/y

map.plot(x, y, 'ro', markersize=10) # Plot city location

plt.text(x, y, city, fontsize=12, ha='right')

# Step 5: Customize the Map

plt.title('City Locations in the USA')

# Step 6: Display the Map

plt.show()