

**NAGARJUNA COLLEGE OF ENGINEERING AND TECHNOLOGY**

**Department of CSE (Data Science)**

**DATA WRANGLING LAB MANUAL**

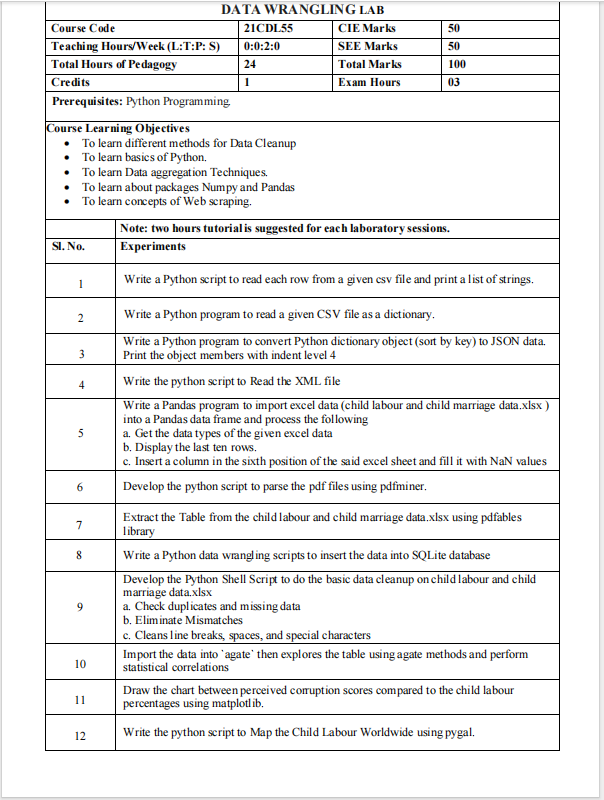
**21CDL55**

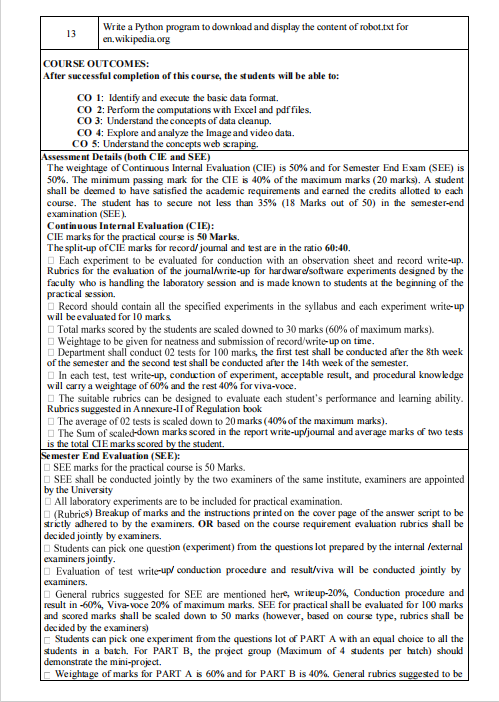
**PREPARED BY**

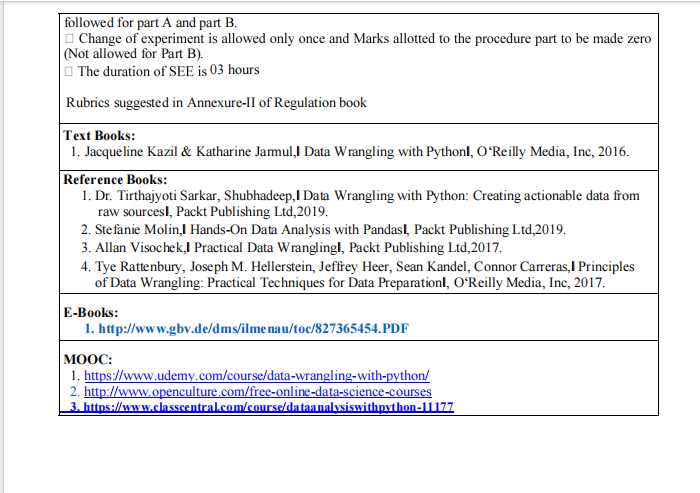
R.RADHAKRISHNAN.,M.E.,(PhD)

ASSISTANT PROFESSOR

Department of CSE (Data Science)







**Ex. 1. Write a Python script to read each row from a given csv file and print a list of strings**

CSV stands for Comma Separated Values. This file format is a delimited text file that uses a comma as a delimiter to separate the text present in it. Or in other words, CSV files are used to store data in tabular form. As per the name suggested, this file contains the data which is separated by a comma and every line of a file is considered a record. We can create the CSV file either from notepad or using excel.

**Using Notepad:**We can create a CSV file using Notepad. In the Notepad, open a new file in which separate the values by comma and save the file with .csv extension.

**Using Excel:**We can also create a CSV file using Excel. In Excel, open a new file in which specify each value in a different cell and save it with filetype CSV.

Using reader we can iterate between rows of a CSV file as a list of values. It iterates over all rows in a CSV file and fetches data in each row as a list. reader() method is present in CSV library. So to use this reader method, first we need to import the CSV library. reader object accepts a single parameter called fileObject (a variable that holds the CSV file).

**Step 1:**In order to read rows in Python, First, we need to load the CSV file in one object. So to load the csv file into an object use open() method.

with open('filename') as fileObject

While loading the file by specifying path along with filename, if you got any unicode error then append r before path of filename

with open(r'path/filename') as fileObject

**Step 2:** Create a reader object by passing the above-created file object to the reader function.

reader\_obj = csv.reader(file\_obj)

**Step 3:** Use for loop on reader object to get each row.

import csv

# Replace 'your\_file.csv' with the path to your CSV file

csv\_file = 'C:\\Users\\errad\\Downloads\\samplecsv.csv'

try:

with open(csv\_file, 'r', newline='') as file:

csv\_reader = csv.reader(file)

for row in csv\_reader:

# Print each row as a list of strings

print(row)

except FileNotFoundError:

print(f"File '{csv\_file}' not found.")

except Exception as e:

print(f"An error occurred: {e}")

Sample I/P

Create a file “samplecsv.csv”. This file contains the following data:

Id,Name,Rating

1,Akhil,4

2,Babu,3

3,Nikhil,5

SAMPLE O/P

['Id', 'Name', 'Rating']

['1', 'Akhil', '4']

['2', 'Babu', '3']

['3', 'Nikhil', '5']

Ex .2 Write a Python program to read a given CSV file as a dictionary.

When using a reader() method we can iterate over a CSV file as a list but using the DictReader class object we can iterate over a CSV file row by row as a dictionary. This DictReader method is present in the csv library. So to use it first we need to import the csv library. DictReader() accepts a single parameter called fileObject (a variable that holds the csv file).

**Syntax**

csv.DictReader(fileobject)

**Step 1:**Load the CSV file using the open method in a file object.

with open('filename') as fileObject

**Step 2:** Create a reader object with the help of DictReader method using fileobject.

reader\_obj = csv.DictReader(file\_obj)

This reader object is also known as an iterator can be used to fetch row-wise data.

**Step 3:** Use for loop on reader object to get each row.

In the code, first we loaded the CSV file named samplecsv.csv and then created a reader\_object that can be iterated to fetch each row. The returned result is in the form of Key-Value pair indicates it as a dictionary. So using DictReader we read data row by row as a Dictionary.

import csv

csv\_file = 'C:\\Users\\errad\\Downloads\\username.csv'

try:

with open(csv\_file, 'r', newline='') as file:

csv\_reader = csv.DictReader(file)

for row in csv\_reader:

print(row)

except FileNotFoundError:

print(f"File '{csv\_file}' not found.")

except Exception as e:

print(f"An error occurred: {e}")

Create a file “samplecsv.csv”. This file contains the following data:

Id,Name,Rating

1,Akhil,4

2,Babu,3

3,Nikhil,5

SAMPLE O/P

{'Id,Name,Rating': '1,Akhil,4'}

{'Id,Name,Rating': '2,Babu,3'}

{'Id,Name,Rating': '3,Nikhil,5'}

**Ex. 3. Write a Python program to convert Python dictionary object (sort by key) to JSON data. Print the object members with indent level 4**

JSON stands for JavaScript Object Notation. It means that a script (executable) file which is made of text in a programming language, is used to store and transfer the data. Python supports JSON through a built-in package called json. To use this feature, we import the JSON package in Python script. The text in JSON is done through quoted-string which contains a value in key-value mapping within { }. It is similar to the dictionary in Python.

**Syntax:**json.dumps(dict, indent)  
**Parameters:** 

* **dictionary –** name of dictionary which should be converted to JSON object.
* **indent –** defines the number of units for indentation

**import** json

# Data to be written

dictionary **=**{

  "id": "04",

  "name": "sunil",

  "department": "HR"

}

# Serializing json

json\_object **=** json.dumps(dictionary, sort\_keys=True, indent **=** 4)

print(json\_object)

Sample o/p

{

"id": "04",

"name": "sunil",

"department": "HR"

}

To read a dictionary and write the o/p in a JSON file

**Syntax:** json.dump(dict, file\_pointer)  
**Parameters:**

* **dictionary –** name of dictionary which should be converted to JSON object.
* **file pointer –** pointer of the file opened in write or append mode.

import json

# Data to be written

dictionary **=**{

    "name" : "sathiyajith",

    "rollno" : 56,

    "cgpa" : 8.6,

    "phonenumber" : "9976770500"

}

with open("sample.json", "w") as outfile:

json.dump(dictionary, outfile)

Sample o/p

{"name": "sathiyajith", "rollno": 56, "cgpa": 8.6, "phonenumber": "9976770500"}

**Ex. 4. Write the python script to Read the XML file**

XML stands for Extensible Markup Language. It's needed for keeping track of the tiny to medium amount of knowledge. It allows programmers to develop their own applications to read data from other applications. The method of reading the information from an XML file and further analyzing its logical structure is known as Parsing. Therefore, reading an XML file is that the same as parsing the XML document.

## Read XML File Using MiniDOM

It is Python module, used to read XML file. It provides parse() function to read XML file. We must import Minidom first before using its function in the application

xml.dom import minidom.parse(filename [,parser[,bufsize]])

from xml.dom import minidom

# parse an xml file by name

file = minidom.parse('C:\\Users\\errad\\Downloads\\ex4.xml')

#use getElementsByTagName() to get tag

models = file.getElementsByTagName('model')

# one specific item attribute

print('model #2 attribute:')

print(models[1].attributes['name'].value)

# all item attributes

print('\nAll attributes:')

for elem in models:

print(elem.attributes['name'].value)

# one specific item's data

print('\nmodel #2 data:')

print(models[1].firstChild.data)

print(models[1].childNodes[0].data)

# all items data

print('\nAll model data:')

for elem in models:

print(elem.firstChild.data)

**Sample I/p**

Save it as ex4.xml

<data>

<models>

<model name="model1">model1abc</model>

<model name="model2">model2abc</model>

</models>

</data>

**Sample o/p**

model #2 attribute:

model2

All attributes:

model1

model2

model #2 data:

model2abc

model2abc

All model data:

model1abc

model2abc

**Ex. 5.Write a Pandas program to import excel data (child labour and child marriage data.xlsx ) into a Pandas data frame and process the following**

**a. Get the data types of the given excel data**

**b. Display the last ten rows.**

**c. Insert a column in the sixth position of the said excel sheet and fill it with NaN values**

import pandas as pd

# Load the Excel file into a Pandas DataFrame

file\_path = "C:\\Users\\errad\\Downloads\\test.xls"

df = pd.read\_excel(file\_path)

# a. Get the data types of the given excel data

data\_types = df.dtypes

print("Data Types:")

print(data\_types)

last\_ten\_rows = df.tail(10)

print("\nLast Ten Rows:")

print(last\_ten\_rows)

# c. Insert a new column with NaN values in the sixth position

new\_column\_name = "New\_Column1"

new\_column\_values = [float('nan')] \* len(df)

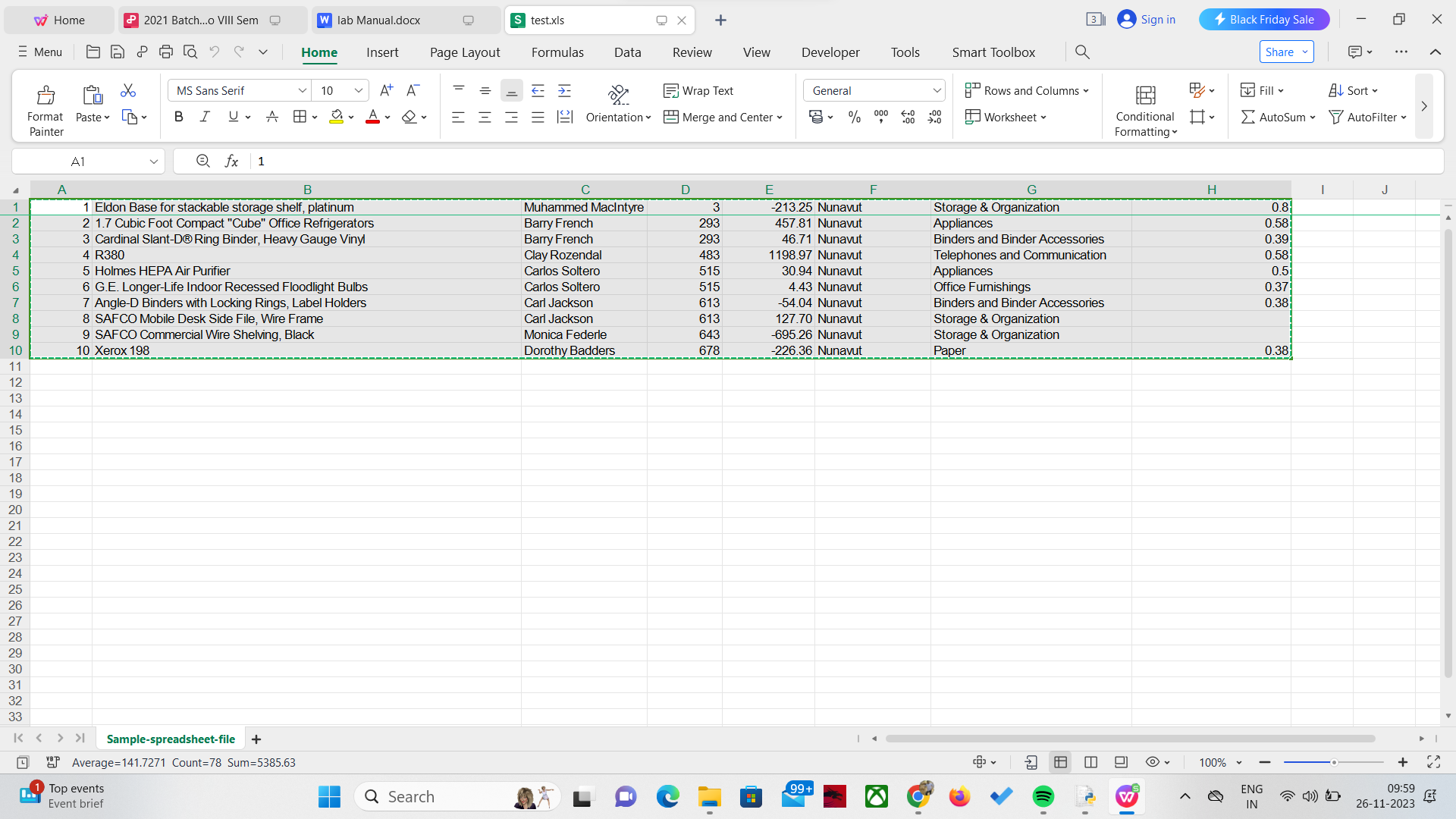
df.insert(5, new\_column\_name, new\_column\_values)

data\_types = df.dtypes

print("Data Types:")

print(data\_types)

SAMPLE I/P



SAMPLE O/P

Data Types:

1 int64

Eldon Base for stackable storage shelf, platinum object

Muhammed MacIntyre object

3 int64

-213.25 float64

Nunavut object

Storage & Organization object

0.8 float64

dtype: object

Last Ten Rows:

1.0 ... 0.8

0 2 ... 0.58

1 3 ... 0.39

2 4 ... 0.58

3 5 ... 0.50

4 6 ... 0.37

5 7 ... 0.38

6 8 ... NaN

7 9 ... NaN

8 10 ... 0.38

[9 rows x 8 columns]

Data Types:

1 int64

Eldon Base for stackable storage shelf, platinum object

Muhammed MacIntyre object

3 int64

-213.25 float64

New\_Column1 float64

Nunavut object

Storage & Organization object

0.8 float64

**EX.No 6. Develop the python script to parse the pdf files using pdfminer.**

from pdfminer.high\_level import extract\_text

def parse\_pdf(pdf\_path):

try:

with open(pdf\_path, 'rb') as file:

text = extract\_text(file)

return text

except Exception as e:

print(f"Error: {e}")

return None

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

pdf\_path = "C:\\Users\errad\\Downloads\\p1.pdf" # Replace with the path to your PDF file

result = parse\_pdf(pdf\_path)

if result:

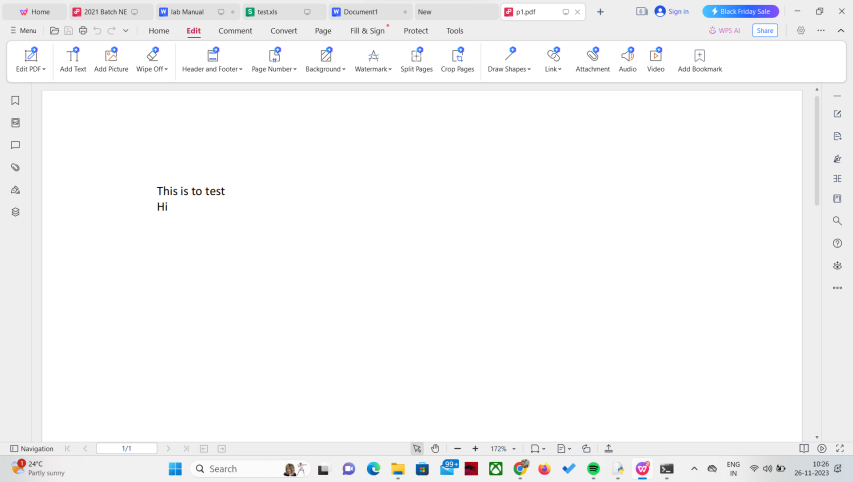
print("Parsed text from PDF:")

print(result)

else:

print("Failed to parse PDF.")

SAMPLE I/P



SAMPLE O/P

Parsed text from PDF:

This is to test

Hi

**Ex No.7.Extract the Table from the child labour and child marriage data.xlsx using pdfables library**

Make sure that Java is installed in the system and environment variable set with java path. Install tabula-py and JPype1 using pip .

#This line imports the **read\_pdf** function from the **tabula** module. This function is used to extract tables f#rom PDF files.

from tabula import read\_pdf

# Function to extract tables from PDF using tabula-py

def extract\_table\_from\_pdf(pdf\_path):

try:

# Using tabula to read tables from PDF

tables = read\_pdf(pdf\_path, pages='all', multiple\_tables=True)

return tables

except Exception as e:

print(f"Error extracting table from PDF: {e}")

return None

This defines a function named **extract\_table\_from\_pdf** that takes a PDF file path as an argument. Inside the function:

It attempts to use the **read\_pdf** function from **tabula** to extract tables from the specified PDF (**pdf\_path**).

The **pages='all'** parameter indicates that tables should be extracted from all pages of the PDF.

The **multiple\_tables=True** parameter indicates that it should attempt to extract multiple tables from each page.

If the extraction is successful, the function returns the extracted tables. If there is an exception during the extraction process, it catches the exception, prints an error message, and returns **None**

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

# PDF file path

pdf\_path = "C:\\Users\\errad\\Downloads\\EN-FINAL Table 9.pdf"

# Extract tables from PDF

pdf\_tables = extract\_table\_from\_pdf(pdf\_path)

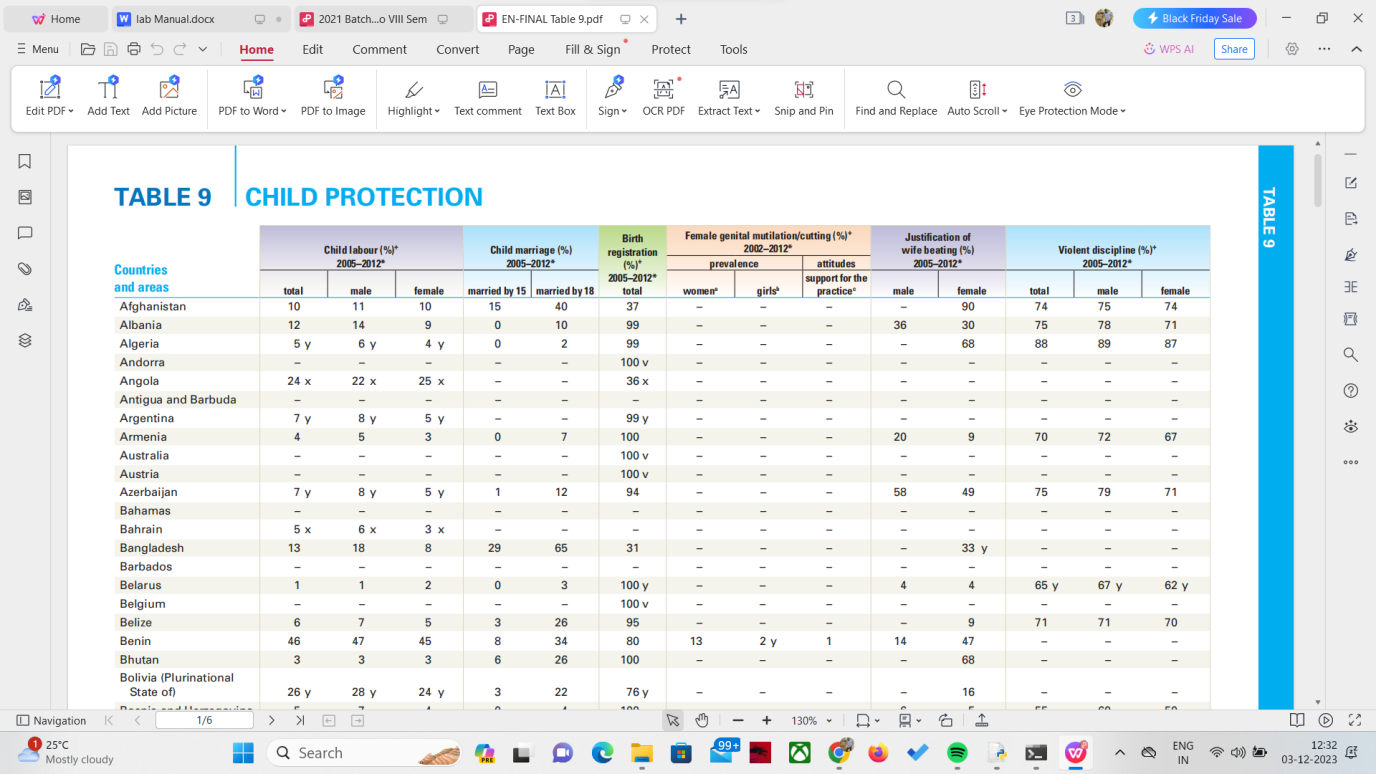
if pdf\_tables:

print("Tables extracted from PDF:")

for i, table in enumerate(pdf\_tables):

print(f"\nTable {i + 1}:\n{table}")

Sample I/P



Sample O/P

Tables extracted from PDF:

Table 1:

Child labour (%)+ Child marriage (%) ... Unnamed: 6 Unnamed: 7

0 2005–2012\* 2005–2012\* ... NaN NaN

1 total male ... male female

2 101110 1540 ... NaN NaN

3 5 y6 y4 y 02 ... NaN NaN

4 24 x22 x25 x –– ... NaN NaN

5 7 y8 y5 y –– ... NaN NaN

6 ––– –– ... NaN NaN

7 7 y8 y5 y 112 ... NaN NaN

8 5 x6 x3 x –– ... NaN NaN

9 ––– –– ... NaN NaN

10 ––– –– ... NaN NaN

11 464745 834 ... NaN NaN

12 26 y28 y24 y 322 ... NaN NaN

13 9 y11 y7 y –– ... NaN NaN

14 ––– –– ... NaN NaN

15 394236 1052 ... NaN NaN

16 3 x,y4 x,y3 x,y 318 ... NaN NaN

17 424340 1338 ... NaN NaN

18 292730 2968 ... NaN NaN

19 3 x3 x2 x –– ... NaN NaN

20 13 y17 y9 y 623 ... NaN NaN

[21 rows x 14 columns]

Table 2:

Child labour (%)+ Child marriage (%) ... Unnamed: 6 Unnamed: 7

0 2005–2012\* 2005–2012\* ... NaN NaN

1 total male ... male female

2 ––– –– ... NaN NaN

3 262528 1033 ... NaN NaN

4 ––– 940 ... NaN NaN

5 ––– –– ... NaN NaN

6 NaN NaN ... NaN NaN

7 151317 939 ... NaN NaN

8 888 25 ... NaN NaN

9 13188 1241 ... NaN NaN

10 9 y14 y4 y 217 ... NaN NaN

11 28 x28 x28 x –– ... NaN NaN

12 ––– –– ... NaN NaN

13 ––– –– ... NaN NaN

14 ––– –– ... NaN NaN

15 192118 736 ... NaN NaN

16 ––– –– ... NaN NaN

17 ––– –– ... NaN NaN

18 26 y35 y16 y 730 ... NaN NaN

19 384036 722 ... NaN NaN

20 242524 318 ... NaN NaN

21 16 x16 x15 x 834 ... NaN NaN

[22 rows x 14 columns]

**Ex.no 8 Write a Python data wrangling scripts to insert the data into SQLite database**

import sqlite3

# Connect to the SQLite database (it will be created if it doesn't exist)

conn = sqlite3.connect('C:\\Users\\errad\\Downloads\\example.db')

# Create a cursor object to execute SQL commands

cursor = conn.cursor()

# Create a table if it doesn't exist

cursor.execute('''

CREATE TABLE IF NOT EXISTS example\_table (

id INTEGER PRIMARY KEY,

name TEXT,

age INTEGER

)

''')

# Sample data to be inserted

data\_to\_insert = [

(1, 'Pranav', 25),

(2, 'Ansari', 30),

(3, 'Binu', 22)

]

# Insert data into the table

cursor.executemany('''

INSERT INTO example\_table (id, name, age) VALUES (?, ?, ?)

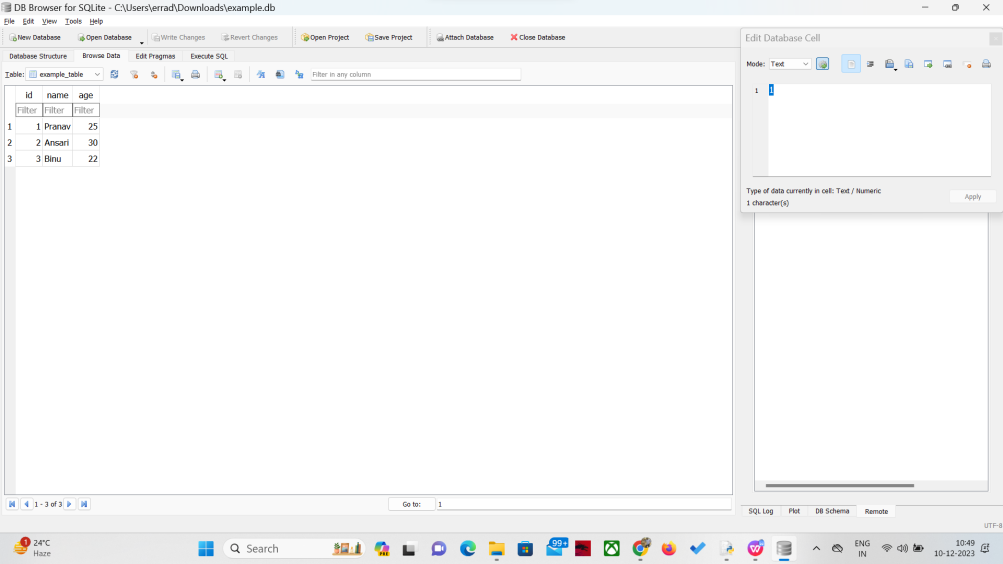
''', data\_to\_insert)

# Commit the changes and close the connection

conn.commit()

conn.close()

Sample o/p



**Ex.9 Develop the Python Shell Script to do the basic data cleanup on child labour and child**

**marriage data.xlsx**

**a. Check duplicates and missing data**

**b. Eliminate Mismatches**

**c. Cleans line breaks, spaces, and special characters**

StepsImport Necessary Libraries:

1. Import the pandas library, which is commonly used for data manipulation and analysis.
2. Define the clean\_data Function:

Create a function named clean\_data that takes the file path of the Excel dataset as an argument.

1. Read the Excel File:

Use pd.read\_excel to read the Excel file into a pandas DataFrame (df).

1. Check for Duplicates:

Use df.duplicated() to identify duplicate rows in the DataFrame.

Print the duplicate rows if they exist; otherwise, print a message indicating no duplicates.

1. Check for Missing Data:

Use df.isnull().sum() to count missing values in each column.

Print the columns with missing data if any; otherwise, print a message indicating no missing data.

1. Eliminate Mismatches:

Clean specific columns (e.g., 'age' and 'gender') based on predefined conditions.

In this example, 'age' is checked to be between 0 and 100, and 'gender' is converted to lowercase.

1. Clean Line Breaks, Spaces, and Special Characters:

Apply the strip() method to remove leading and trailing whitespaces.

Use applymap() to apply the stripping operation to all columns.

1. Save the Cleaned DataFrame to a New Excel File:

Use df.to\_excel to save the cleaned DataFrame to a new Excel file named "cleaned\_data.xlsx".

1. Print a Message Indicating the Cleaning Process is Complete:

Display a message indicating that the cleaned data has been saved to a new file.

1. Specify the Path to the Excel File and Execute the Cleaning Process:

Specify the path to your original Excel file (your\_dataset.xlsx) and call the clean\_data function to execute the cleaning process.

import pandas as pd

def clean\_data(file\_path):

# Read the Excel file into a pandas DataFrame

df = pd.read\_excel(file\_path)

# a. Check duplicates and missing data

duplicates = df[df.duplicated()]

missing\_data = df.isnull().sum()

if not duplicates.empty:

print("Duplicate Rows:")

print(duplicates)

else:

print("No duplicate rows found.")

if not missing\_data.empty:

print("\nMissing Data:")

print(missing\_data)

else:

print("No missing data found.")

# b. Eliminate Mismatches (customize this based on your columns)

# Example: Assuming 'age' and 'gender' columns need to be cleaned

df['age'] = df['age'].apply(lambda x: x if 20 < x < 100 else pd.NA)

# Handle NaN values before applying lower()

df['gender'] = df['gender'].apply(lambda x: x.lower() if isinstance(x, str) else pd.NA)

# c. Clean line breaks, spaces, and special characters (apply to all columns)

df = df.applymap(lambda x: x.strip() if isinstance(x, str) else x)

df = df.applymap(lambda x: x.replace('@', '') if isinstance(x, str) else x)

# Save the cleaned DataFrame back to the Excel file

cleaned\_file\_path = "C:\\Users\\errad\\Downloads\\cleaned\_data.xlsx"

df.to\_excel(cleaned\_file\_path, index=False)

print(f"\nCleaned data saved to {cleaned\_file\_path}")

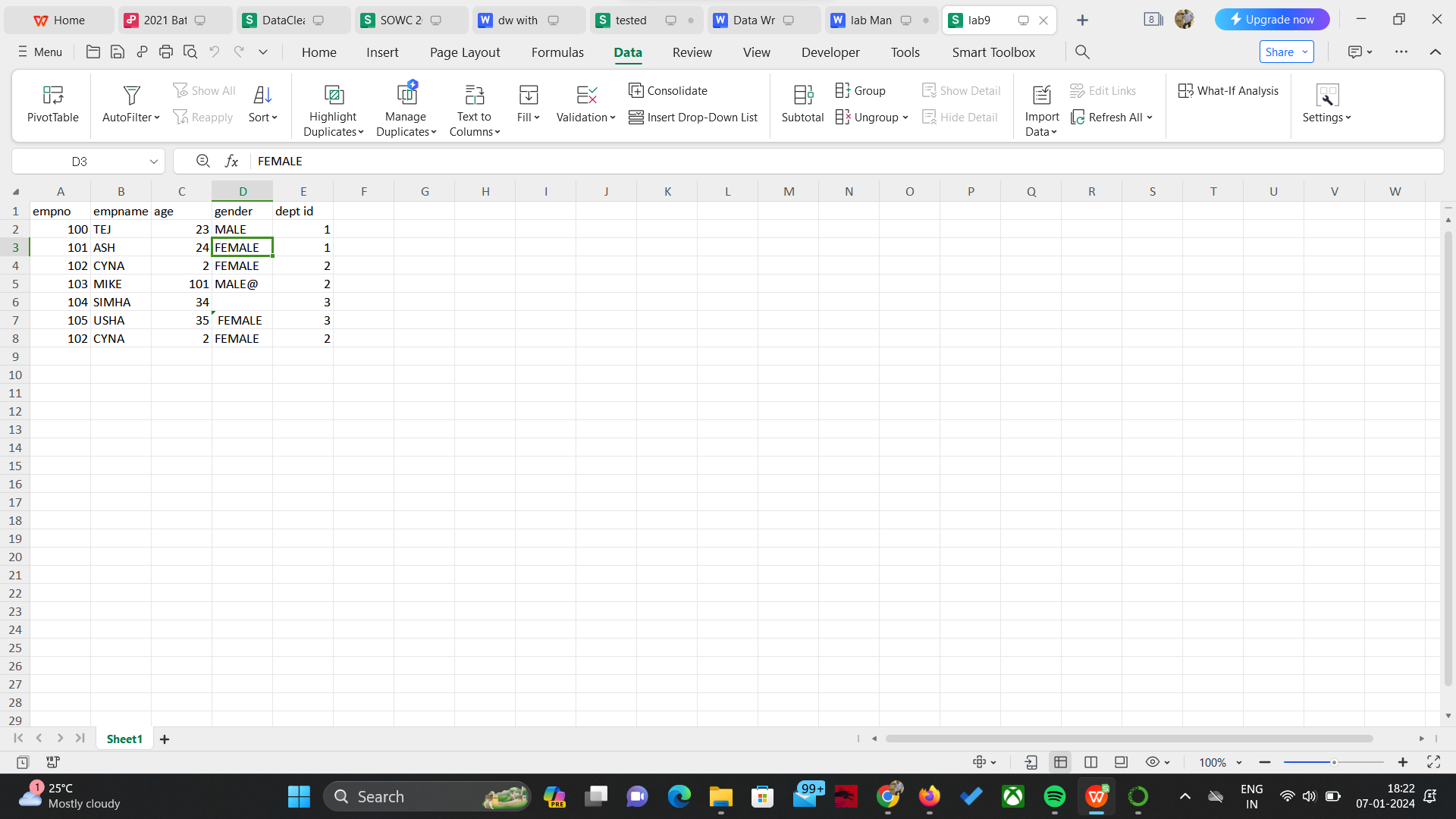
# Specify the path to your Excel file

file\_path = "C:\\Users\\errad\\Downloads\\lab9.xlsx"

# Perform data cleanup

clean\_data(file\_path)

Sample I/p



Sample o/p

