Department of Artificial Intelligence and Machine Learning

III B.Tech I Sem

DATA WRANGLING AND PRE- PROCESSING

UNIT-1

Introduction to Data Wrangling

By

Mr.N.Siva,

Assistant Professor,

Department of AI&ML.

Data Wrangling and Preprocessing

UNIT-1: Introduction to Data Wrangling

What is Data wrangling, Importance of DW, Tasks of Data wrangling, Data Wrangling Tools, Introduction to python for Data Wrangling, Python basics for Data wrangling, Handling Structured data: CSV, JSON, and XML formats, Data Meant to Be Read by Machines.

What is Data Wrangling ?

Data Wrangling (also known as Data Mugging) is the process of cleaning, structuring, enriching, validating and transforming raw data into a described format for better decession making and analysis.

1. cleaning : Handling missing & Duplicate data.

cleaning is the process of handling missing, incorrect, inconsistent or duplicate data to improve data quality.

```
import pandas as pd

data = {
    'Name' : ['John', 'Mary', 'sam', 'Mary'],
    'Age' : [25, None, 30, None],
    'Salary' : ['$5000', '$6000', '7000', '$6000']
}
df = pd.DataFrame(data)  → print("original Data")
                           print(df)

#clean step
df = df.drop_duplicate()  #Remove duplicate rows.
df['Age'] = df['Age'].fillna(df['Age'].mean())
                          # fill missing ages with average
df['salary'] = df['salary'].str.replace('$', '').
                astype(float)
                          #Remove $ and convert to float.
```

Original Data

	Name	Age	Salary
0	John	25.0	$5000
1	Mary	NaN	$6000
2	Sam	30.0	7000
3	Mary	Nan	$6000

cleaned Data

	Name	Age	Salary
0	John	25.0	5000.0
1	Mary	27.5	6000.0
2	Sam	30.0	7000.0

2 structuring : Split and organize Data.

Structuring means converting unorganized or semi structured data into a well-organized format, such as rows and columns.

```
import pandas as pd
#Raw unstrured data
data = {
    'info': ['John, 25, 5000',
             'Mary, 30, 6000',
             'Sam, 28, 4500']
}
df = pd.DataFrame(data)
print("original Data")
print(df)
```

Split the 'Info' column into Structured Columns

```
df[['Name', 'Age', 'salary']] = df['Info'].str.
                                split(',', expand=True)

df.drop('Info', axis=1, inplace=True)
print("\n structured Data")
print(df).
```

original Data

```
0  John,25,5000
1  Mary,30,6000
2  sam,28,4500
```

structured data

	Name	Age	salary
0	John	25	5000
1	Mary	30	6000
2	sam	28	4500

Note:

→ we want to split the "Info" column into three seperate columns.

Name Age Salary

```
df['Info'].str.split(',', expand=True)
```

→ This splits each string in the Info column by comma ,

→ expand=true means, create seperate columns for the split parts

```
df.drop('Info', axis=1, inplace=True)
```

→ This removes original Info column

→ axis=1 means we are dropping a column (not a row)

→ inplace=True means the change is applied directly to df.

3. Enriching: Add New Columns.

Enriching involves adding or creating new data fields from existing data to provide more insights.

```
import pandas as pd

data = {
    'Name': ['John', 'Mary', 'Sam'],
    'Age': [25, 30, 28],
    'Salary': [5000, 6000, 4500]
}
df = pd.DataFrame(data)
# Add New Columns for Tax
df['Tax'] = df['Salary'] * 0.20
# Add New Columns for Age Category
df['Category'] = df['Age'].apply(lamda x:
                    'young' if x<30 el
                    'Adult')

print("Enriched data")
print(df)
```

Enriched data

	Name	Age	Sal	Tax	Categ
0	John	25	5000	1000.0	Youn
1	Mary	30	6000	1200.0	Adu
2	Sam	28	4500	900.0	Youn

4 Validating : Check for Errors and Inconsistencies

Validating ensures that data is accurate, consistant and meets required formats or rules.

```
import pandas as pd

data = {
    'Name': ['John', 'mary', 'sam'],
    'Age': [25, -30, 28], # Invalid negative Age
    'Salary': [5000, -6000, 4500]
                # invalid Salary negative Salary
}

df = pd.DataFrame(data)
print("Data for Validation")
print(df)

#check for Negative Salary
print("\n Negative salaries")
print(df[df['Salary']<0])
#Check for Negative Age
print("\n Negative Age")
print(df[df['Age']<0])
```

Data for Validation

	Name	Age	Salary
0	John	25	5000
1	Mary	-30	-6000
2	Sam	28	4500

Negative salaries

	Name	Age	salary
1	mary	-30	-6000

Negative Age

	Name	Age	Salary
1	mary	-30	-6000

Drop rows where Age or salary is negative

```
df = df [ (df ['Age'] >= 0) & (df ['salary'] >= 0)]

print ("\n cleaned data")
print (df)
```

Cleaned data

	Name	Age	salary
0	John	25	5000
2	Sam	28	4500

5. Transforming : change format or unit.

Transforming involves converting data formats or units to meet analysis or reporting requirements.

```
import pandas as pd
data = {
    'Name' : ['John', 'Mary', 'Sam'],
    'JoinDate' : ['01/06/2023', '15/07/2022',
                  '10/08/2023'],
    'Salary_USD' : [5000, 6000, 4500].
}
df = pd.DataFrame(data)
# convert joinDate from string to date time format
df['JoinDate'] = pd.to_datetime(df['JoinDate'],
                                format = '%d/%m/%Y')
# convert Salary from USD to INR (1 USD = 83.2 INR)
    conversion_rate = 83.2
df['salary_INR'] = df['salary_USD'] * conversion_rate.
    print("Transformed Data :")
    print(df).
```

Transformed Data:

Name	Join Date	salary_USD	salary_INR
John	2023-06-01	5000	416000.0
Mary	2022-07-15	6000	499200.0
Sam	2023-08-10	4500	374400.0

2. Importance of Data wrangling:

1. Improves Data Quality: Raw data often contain inconsistencies, duplicates or missing values. wrangling fixes thes issues.

Ex: Sales dataset

Customer Name	purchase Date	Amount
John	12/05/2024	500
John	12-05-2024	500
Smith	NULL	650

issues:

→ Duplicate rows for John
→ Different date formats (12/05/2024 Vs 12-05-2024)
→ missing purchase Data

Data wrangling fixes:

→ Removes duplicates

→ Standardizes date format (YYYY-MM-DD)

→ Fill or flags missing data

2. Save Time:

Data scientists often spend 70-80% of their time ~~for~~ preparing data (process of transforming raw, messy, incomplete data into a clean and structured format) for analysis, visualization or machine learning.

Instead of manully cleaning 10,000 rows in Excel A python script using pandas can clean the entire file in seconds.

3. Enhance Decession-Making: cleane and structured data leads to more accurate analysis and insights.

4. Supports Machine Learning: Machine learning algorithms require clean, formatted, and normalized data to perform well.

5. Integrates Diverse Sources: combines data from different formats, systems, or databases for a unified view.

Tasks of Data Wrangling :

Data wrangling involves multiple tasks that help convert raw, unstructured or messy data into clean and structured format suitable for analysis or machine learning.

main Tasks of Data wrangling :

1. Data collection
2. Data cleaning
3. Data Transformation
4. Data Integration
5. Data Reduction
6. Data Validation
7. Data Enrichment

1. Data collection :

Gathering data from various sources like csv files, databases, APIs or web scrapping.

```
import pandas as pd
df = pd.read_csv ("employees.csv")
                    # Load data from csv
```

2. Data cleaning:

Fixing or removing incorrect, corrupted, missing or duplicate data.

Common cleaning Tasks:

- → Handling missing values
- → Fixing wrong data types
- → Removing duplicates
- → Correcting Types

```
df.drop_duplicates(inplace=True)
df['Age'].fillna(df['Age'].mean(),
                 inplace=True)
```

3. Data Transformation:

Changing the format, structure or values of data to make it more usable.

```
df['salary_INR'] = df['salary_USD'] * 83.2  # convert currency
df['Gender'] = df['Gender'].map({'Male':1,
                                 'Female':0})
# Encode gender.
```

4. Data Integration

combining data from multiple sources into a single dataset

Ex:

```
df1 = pd.read_csv('employee.csv')
df2 = pd.read_csv('department.csv')
df_combined = pd.merge(df1, df2, on='EmpID')
```

5. Data Reduction

Reducing the Volume of data by removing irrelevant or redundant information.

```
df = df[['EmpId', 'Name', 'Salary']]
                    # Keep only important columns
```

6 Data Validation:

Ensures that the data meets Specific quality rules (Eg. no negative ages or salaries)

```
df = df[(df['Age']>0) & (df['salary']>0)]
```

7) Data Enrichment:

Enriching data by adding new relevant information (from external or derived sources)

df['experience'] = 2025 - df['Joining Year']

calculation of Experience

Introduction to python for Data wrangling:

What is python:

Python is a high-level, interpreted and general purpose programming language known for its simplicity, readability and strong community support. It is one of the most widely used languages in data science, machine learning and data wrangling

Why use python for Data wrangling:

-> Easy to learn and use

-> Rich set of libraries (pandas, Numpy etc)

-> Supports multiple data formats (csv, Excel, JSON, XML)

-> Scalable and works with large datasets.

-> Integrates well with visualization tools and databases

Key Libraries for Data Wrangling in Python:

Library	purpose
Pandas	Data manipulation & Analysis
Numpy	Numerical operations & Arrays
Openpyxl, xlrd	Read/write Excel files
json	Work with jasn data
csv	Read/write csv files
re	Regular Expressions (for text claning)

Basic workflow of Data wrangling in Python:

1. import data
2. inspect and understand data
3. clean data (remove missing values, duplicates etc)
4. Transform data (rename columns, change types etc)
5. Export cleaned data

```
import pandas as pd
# step1: Load data from csv file
df = pd.read_csv ('employees.csv')
# step2: view first 5 rows.
print (df.head())
# step3: check for missing values
print (df.isnull().sum())
# step4: Drop rows with missing values
df_clean = df.dropna()
#step5: Rename columns
df_clean.rename(columns={'EmpName': 'Employee_Name'},
                inplace=True)
# step6: save cleaned data
df_clean.to_csv ('cleaned_employees.csv',
                 index = False)
```

Common Python Data Wrangling Tasks:

Task	Function / Method
Load csv	`pd.read_csv()`
Drop missing values	`df.dropna()`
Fill missing values	`df.fillna(value)`
Remove duplicates	`df.drop_duplicates()`
filter rows	`df[df['Age']>30]`
Rename columns	`df.rename(columns={'old':'New'})`
change data types	`df['Age'] = df['Age'].astype(int)`
String operations	`df['Name'].str.upper()`

10/7/25: new R3-03 present.

```
import pandas as pd
# step1: Read the original data
df = pd.read_csv('c:/user/ .... /employees.csv')
# step2. clean - Remove rows with missing salary
df_clean = df[df['salary'].notnull()].copy()
# step3: clean - Rename column
df_clean.rename(columns={'EmpName':
                         'Employee_Name'},
                inplace=True)
# step4: Drop duplicates.
df_clean.drop_duplicate(inplace=True)
# step5: Save cleaned data
df_cleaned.
df_clean.to_csv('c:/user/.... /cleaned_emp.csv',
                index=False)
print("cleaned data saved successfully").
```

Python basics for Data Wrangling:

1. Variables and Data types:

A variable is a name that refers to a value stored in memory.

Data type define the type of value a variable holds, such as numbers, text, or logical values.

Common python data types:

int - Integer numbers (eg: 10)

float - Decimal numbers (eg: 10.5)

Str - String (text) data (eg: "Data")

bool - Boolean (True or False)

Example:

```
name = "Siva"      #str
age = 25           #int
height = 5.7       #float
is_student = True  #bool
```

2. Lists:

A list is a mutable, ordered collection of values.

List allow you to store multiple values in a single variable. You can change, add or remove items.

Ex:

```
fruits = ["apple", 'banana', 'mango']
print(fruits[1])    # output: banana
fruits.append("grape")  # Add a new item
```

3. Dictionaries

A dictionary is an unordered collection of key-value pairs. Each key maps to a specific value. It is useful for structured data (like JSON)

```
Student = {
        "Name": "Raju",
        "age": 25,
        "branch": "AIML"
}
print(student["branch"])  # output: AIML
```

4. Control flow (if, for, while)

Control flow statements manage the execution path of a program.

if: executes code based on condition

for: repeats over items

while: loops while condition is true.

#if

```
Score = 80
if score > 50:
    print("Pass")
```

Pass

for loop

```
for i in range(3):
    print("Hello", i)
```

Hello 0
Hello 1
Hello 2

while loop

```
Count = 0
while count < 3:
    print("count:", count)
    Count += 1
```

count 0
count 1
count 2

5. Functions:

A function is a reusable block of code that performs a specific task. We can define our own functions with "def", and call them using their name.

Ex:
```
def greet(name):
    return "Hello," + name
print(greet("Raj"))  # output : Hello, Raj
```

6. File Handling:

File handling refers to reading from and writing to files. Files can be opened using open() with modes like r (read), w (write) and a (append).

Ex:
```
# write to a file
with open("demo.txt","w") as f:
    f.write("Welcome to Python")

# Read from a file
with open("demo.txt","r") as f:
    print(f.read())
```

7 Importing libraries:

Importing allows you to use external python libraries. Use "import" to bring in built-in or third party modules like pandas, numpy, etc.

Example:

```
import pandas as pd
import numpy as np
import json.
```

8) Reading csv using pandas:

CSV (Comma Seperated Values) is a file format used to store tabular data.

Ex:

```
import pandas as pd
df = pd.read_csv("Employees.csv")
print(df.head())
```

9) working with JSON Data:

JSON (Java Script Object Notation) is a format for storing and exchanging data. Use the json module to load or write JSON data

```
import json
with open ("data.json") as f:
    data = json.load (f)
for emp in data ["employees"]:
    print (emp ["name"], emp ["department"])
```

10) Regular Expression (re module)

Regular expressions are used to match text patterns. Python's re module allows pattern matching and text replacement

Ex:

```
import re
text = "phone: 9876543210"
match = re.search (r"\d{10}", text)
if match:
    print ("Found number:", match.group())
```

4. Numpy Basics:

Numpy is a python library for numerical and array operations.

```
import numpy as py
arr = np.array[1,2,3,4,5]
print (arr.mean())   # output : 3.0
```

Handling structured data: csv, json, xml

csv is a plain text format where each line represents a data record. Each field is seperated by a comma.

* why use csv?

→ Easy to read and write
→ Common format for tabular data.
→ Supported by Excel and databases.

employees.csv

```
EmpID, EmpName, Department, Salary
101, Alice, HR, 50000
102, Bob, IT, 60000
103, Charlie, Finance, 55000
```

```
import pandas as pd
#Load csv
df = pd.read_csv('employees.csv')
#Display the data frame
print(df)
#Access specific columns
print("\n Employee Names:")
print(df['EmpName'])
```

output:

	EmpID	EmpName	Department	Salary
0	101	Alice	HR	50000
1	102	Bob	IT	60000
2	103	charlie	finance	55000

Employee Names:

0 Alice
1 Bob
2 charlie

Name: EmpName, dtype: object

2. JSON (Java Script Object Notation)

JSON is a lightweight data-interchange format. It is structured like a python dictionary and supports nesting.

Sample.json data.json

```
{
  "employees": [
    { "name": "Alice", "department": "HR" },
    { "name": "Bob", "department": "IT" },
    { "name": "Charlie", "department": "Finance" }
  ]
}
```

```
import json

#Load JSON
with open('data.json') as f:
    data = json.load(f)

#Access employee data
print("Employees Details:")
for emp in data['employees']:
    print(emp['name'], emp['department'])
```

output:

Employee Details:

Alice HR
Bob IT
Charlie Finance

3. XML (eXtensible Markup Language)

XML is a markup language that stores data in a structured, hierarchical format using user-defined tags.

Sample XML: employees.xml

```
<company>
  <employee>
  <name>Alice</name>
  <department>HR</department>
  </employee>

  <employee>
  <name>Bob</name>
  <department>IT</department>
  </employee>

  <employee>
  <name>Charlie</name>
  <department>Finance</department>
  </employee>

  </company>
```

```
import xml.etree.ElementTree as ET

# parse XML
tree = ET.parse('employees.xml')
root = tree.getroot()

# Access employee data
print("Employee Data from XML:")
for emp in root.findall('employee'):
    name = emp.find('name').text
    dept = emp.find('department').text
    print(name, dept)
```

output:

```
Employee Data from XML:
Alice HR
Bob IT
Charlie Finance
```

Department of Artificial Intelligence and Machine Learning

III B.Tech I Sem

DATA WRANGLING AND PRE- PROCESSING

UNIT-2

Working with Excel Files, PDFs, and Databases

By

Mr.N.Siva,

Assistant Professor,

Department of AI&ML.

UNIT-II

Working with Excel Files, PDFs and Databases

Installing python Packages for Data Wrangling,

Parsing Excel Files,

Programatic Approaches to PDF parsing

Converting PDF to Text (pdfminer)

Acquiring and Storing Data

Introduction to Databases for Data wrangling

Relational Databases: MySQL and PostgreSQL.

Non Relational Databases: NoSQL and Alternative Data storage.

1. Installing Python Packages for Data wrangling:

Install external libraries that help handle Excel, PDF, and database files.

Common Packages:

* pandas – For structured data (Excel, csv)
* openpyxl, xlrd – For Excel format
* pdfminer.six, PyPDF2 – for PDF parsing
* sqlalchemy, sqlite3 – For SQL databases
* pymongo – for MongoDB (No SQL)

Installation steps:

Step1: Open Terminal or Command prompt

press windows + R and type cmd and press Enter

Step2: Run installation Command

pip install pandas openpyxl xlrd pdfminer.six sqlalchemy pymongo

Step3: wait for installation to Complete then you will see msg like

Successfully installed pandas openxl xlrd ....

now setup is completed.

step4: Verify installation

open a python shell IDLE and press control+N then type below packages.

```
import pandas as pd
import openpyxl
import xlrd
import sqlalchemy
import pymongo
from pdfminer.high_level import extract_text
print("All packages imported Successfully")
```

output:

All packages imported Successfully.

Note: if you are using Python 3.13, pip is not recognized, try: `Py -3.13 -m pip install openpyxl`

2 Parsing Excel files:

Excel is a widely used Spreadsheet format to store structured data in rows and columns. it comes in two main file types.

1. .x/sx : Excel workbook (New format)

-> Introduced in Excel 2007
-> xml based (open xml format)
-> Support over a million rows.
-> Commonly used and recomended
-> can be read using openpyxl

2 .xls : Excel 97-2003 workbook (old format)

-> Binary file format
-> Limited to 65,536 rows
-> can be read using xlrd

Why use python for Excel:

Excel is good for manual entry, but not for automation. Python makes it easier to

-> Read Excel data in bulk
-> Analize, Filter and Sort

→ Add new Columns

→ Save results back to new Excel files

→ Automate repetitive Excel processing tasks

Required Python Libraries:

Library	Purpose
pandas	Read and manipulate data
openpyxl	Read/write .xlsx Excel files
xlrd	Read ~~.xlsx~~ .xls Excel Files

How to install: in Command prompt

```
py -3.13 -m pip install pandas ~~openpyxl~~
                                openpyxl xlrd
```

Read .xlsx File Using openpyxl

```
import pandas as pd
#Read new format Excel files (.xlsx)
df = pd.read_excel('sample.xlsx', engine=
                                'openpyxl')
print(df)
```

* pd.read_excel() : reads Excel files into a pandas DataFrame

* engine='openpyxl' : tells pandas to use the openpyxl library for .xlsx files.

Read .xls Files using xlrd

```
import pandas as pd
# Read old format Excel file (.xls)
df = pd.read_excel('sample.xls', engine='xl
print(df)
```

Common operations :

1) Filter Rows — Get students with score >

```
high_score = df[df['score'] > 80]
print(high_score)
```

2) Sort Data by Score (Desending)

```
Sorted_df = df.sort_values(by='score',
                           ascending=False)
print(sorted_df)
```

3. Add a new Column : Grade

```
df['Grade'] = df['score'].apply(lambda s:
                'A' if s>=90 else
                'B' if s>=80 else 'C')
print(df)
```

4. Save the filtered Data to a New Excel file:

```
high scores.to_excel('high score.xlsx', index=
                                        False)
```

Data.Exlsx

Name	Score
Alice	88
Bob	75
Charlie	95
David	67

Programmatic Approaches to PDF parsing in Python

Parsing PDFs programatically allows automated extraction of data like:

→ Text Content

→ Tables

→ meta data

Python offers powerful libraries for PDF parsing.

→ pdfminer.six → Best for text extraction

→ PyPDF2 or pypdf → Basic text + metadata

→ pdfplumber → for table and structured text data

1. install Required Libraries:

```
pip install pdfminer.six  PyPDF2  pdfplumber
```

2. Parsing PDFs using pdfminer.six (Best for plain text)

* Extract All Text from PDF

```
from pdfminer.high_level import extract_text
#Read all text from PDF
text = extract_text ("sample.pdf")
print (text)
```

* Save Extracted Text to file

```
with open ('sample.txt', 'w', encoding='utf-8')
    as f:
    f.write(text)
```

3 Parsing PDFs using PyPDF2 (or Pypdf)

Basic Text Extraction

PyPDF is a python library that allows you to read, extract, split and merge PDF files. It's useful for basic operations such as extracting text content from pages or merge mutiple PDFs.

Use cases :

→ Extracting text from PDFs
→ Reading pages individually
→ Merging or splitting PDF documents
→ Getting PDF metadata

Installing PyPDF2

pip install PyPDF2

> pip install PyPDF2
> python.exe -m pip install --upgrade pip

Extract text from PDF

Let's say you have a PDF file named "sampl.pdf" with 2 pages.

```
import PyPDF2
# Open PDF File
with open('sample.pdf', 'rb') as file:
read = PyPDF2.PdfReader(file)
# Print total pages
print("Total Pages:", len(reader.pages))
# Extract text from each page
for i, page in enumerate(reader.pages):
    print(f"\n--- Page {i+1} ---")
    print(page.extract_text())
```

Sample Output

If your sample.pdf has the following text:

- page 1: "Welcome to Data Wrangling"
- page 2: "This PDF demonstrates PyPDF2"

output:

Total pages: 2

---page 1---

Welcome to Data Wrangling

---page 2---

This PDF demonstrates ~~PDF~~ PyPDF2

4 (Pdfplumber): Parsing PDFs using pdfplumber:

pdfplumber is a python library designed for extracting text, tables, and metadata from PDF files. It works especially well with PDFs that contain complex layouts or tables, making it a popular choice when the boilt in methods (such as those from PyPDF2) may not extract information correctly.

* installing pdfplumber:

Before running Code, you need to install pdfplumber. Open your Command prompt or terminal and run.

```
pip install pdfplumber
```

```
-3.13 -m -pip install pdfplumber
```

Basic Usage: Extract Text from a PDF:

```
import pdfplumber
# Specify the path to your PDF file
pdf_path = r"C:\User\Sivan\Desktop\sampl.pdf"

# open the PDF using pdfplumber
with pdfplumber.open(pdf_path) as pdf:
    # print total number of pages
    print("Total pages:", len(pdf.pages))
    # Iterate through each page and extract
    for i, page in enumerate(pdf.page
        text = page.extract_text()
        print(f"\n-- page {i+1} --
        print(text)
```

output: sample.pdf

page 1 content

"welcome to pdfplumber"

pag 2 content

"welcome to Data Wrangling class"

output:

Total pages: 2

--- page 1 ---

"welcome to pdfplumber.

--- page 2 ---

welcome to Data Wrangling class

Acquiring and storing Data:

Data acquisition and storage is the foundation of any data wrangling workflow. It refers to the process of "gathering data from various sources and storing it in suitable formats for further processing and analysis.

1. Sources of Data Acquisition:

You can aquire data from the following sources:

Sources	Tools / methods
Files	CSV, Excel, JSON, XML, Text
Databases	MySQL, PostgreSQL, SQLite, MongoDB
Web API	REST APIs using requests, json
Web Scraping	HTML data using BeautifulSoup, Selenium
PDFs	Pdfminer, PyPDF2, pdfplumber
Cloud storage	Google Drive, AWS S3, Firebase etc.

2. Storing formats:

After acquiring data, it can be stored in:

* Structured formats: CSV, Excel, Databases
* Semi_structured formats: JSON, XML
* Unstructured formats: Text, Images, PDF

1. Reading from a csv file and Saving to Excel:

```
import pandas as pd

# Read csv file
df = pd.read_csv("data.csv")

# Display Data
print("csv Data:\n", df.head())

# store to Excel
df.to_excel("ouput_data.xlsx", index=False)

print("Data stored in Excel Successfully")
```

Q. Fetching Data from a REST API and Storing to JSON :

```
import requests
import json
# API Endpoint (Example)
url = "https://jsonplaceholder.typicode.com/posts"
# Send GET request
response = requests.get(url)
data = response.json()
# print first 2 records
print(json.dumps(data[:2], indent=4))
# save data to a JSON file
with open("api_data.json", 'w') as f:
    json.dump(data, f, indent=4)
print("API data Stored to JSON successfully")
```

3. Reading from an Excel file and saving to CSV:

```
import pandas as pd

#Read Excel file
df = pd.read_excel("input_data.xlsx")

#Display first few rows
print(df.head())

# save to csv
df.to_csv("converted_data.csv", index=False)
print("Excel data converted to csv successfully")
```

4. CSV to JSON

```
import pandas as pd

#Read csv
df=pd.read_csv("data.csv")

#convert to JSON and save
df.to_json("converted_data.json",
           orient="records", indent=4)

print("csv converted to JSON successfully")
```

5) Excel to CSV

```
import pandas as pd

#Read Excel
df = pd.read_excel("data.xlsx")
# convert to csv
df.to_csv("converted_data.csv", index=Fa
print("Excel converted to csv Successfully")
```

6) JSON to CSV:

```
import pandas as pd
#Read JSON
df = pd.read_json("data.json")
#Convert to csv
df.to_csv("converted_from_json.csv",
                    index = False)
print("JSON converted to csv Successfully
```

7) JSON to Excel

```
import pandas as pd
df = pd.read_json("data.json")
df.to_excel("converted_from_json.xlsx",
                index=false)
print("JSON converted to Excel Successfully")
```

Introduction to Databases for Data Wrangling:

→ What is Data wrangling:

Data wrangling is the process of:

* collecting data from various sources
* cleaning, transforming and formatting it
* preparing it for analysis or machine Learning

→ Why use Databases in Data wrangling:

Databases helps to manage large and structured data sets in a reliable, scalable and efficient manner.

Benifits:

Features	why it helps in DW
Structured Data	Tables with rows & columns make wrangling easier
Quering with SQL	we can filter, join, group and sort data easily

Efficient Storage — store millions of rows efficiently

Security & Access — control who can view or edit data

Integration — connect with Excel, APIs, scripts and more.

Types of Databases:

Type	Description	Example Tools
Relational (SQL)	Data in tables with fixed schema	MySQL, PostgreSQL, SQLite
Non-Relational (NoSQL)	flexible, JSON-like data	MongoDB, Firebase

Tools Commonly Used with Python:

Tool/Library	purpose
Sqlite3	Connect to SQLite databases
mysql-connector-python	Connect to MySQL
psycopg2	Connect to PostgreSQL
sqlalchemy	unified ORM for databases

pymongo — work with MongoDB (NoSQL)

pandas.read_sql() — Read query results as DataFrames

Using SQLite for Data Wrangling:

Python Code to Create, Insert and Read Table

```
import sqlite3
import pandas as pd

# step1: connect to database (create file if not exists)
conn = sqlite3.connect('student.db')
cursor = conn.cursor()

# step 2: Create table
cursor.execute('''
    CREATE TABLE IF NOT EXISTS students(
        id INTEGER PRIMAR KEY,
        name TEXT,
        grade INTEGER
    )
''')
```

```
# step3: Insert data
cursor.executemany("INSERT INTO
                    students (id, name, grade)
                    VALUES (?,?,?)",
[(1, 'Alice', 85),
 (2, 'Bob', 90),
 (3, 'charlie', 78)
])
conn.commit()
# Step 4: Read data using pandas
df = pd.read_sql("SELECT * FROM
                  students", conn)
print("student Data").
print(df)
conn.close()
```

output

student Data

id	name	grade
1	Alice	85
2	Bob	90
3	Charlie	78

MySQL Version:

① Setup MySQL on your Machine:

→ Download from: https://dev.mysql.com/downloads/

→ Install MySQL server & workbench (optional GUI)

→ During Setup:

* set username: root
* password: ****

② Install Required Python Libraries:

pip install mysql-connector-python pandas openpyxl

→ mysql-connector-python: MySQL Connector

→ pandas: Data manipulation

→ openpyxl: Export to Excel

③ Create Database and Table in MySQL:

use mysql workbench or terminal:

```
CREATE DATABASE customerdb;
USE customerdb;
```

```
CREATE TABLE customers(
id INT AUTO_INCREMENT PRIMARY KEY,
name VARCHAR(100),
email VARCHAR(100),
status VARCHAR(20),
purchase_amount FLOAT
);
```

④ Insert Sample Data (with duplicates)

```
INSERT INTO customers (name, email, status, purchase_
amount) VALUES
('Alice', 'alice@example.com', 'active', 250.50),
('Bob', 'bob@example.com', 'inactive', 0),
('Alice', 'alice@example.com', 'active', 250.50),
('Eve', 'eve@example.com', 'active', 120.75),
('Charlie', 'Charlie@example.com', 'active', 300.00);
```

5. python script (mysql) - wrangling_mysql.py

```
import mysql.connector
import pandas as pd
#step1: connect to ~~sql~~ MySQL
conn = mysql.connector.connect(
host = "local host",
user = "root",
password = "your_password",
database = "customerdb".
).

# step2: clean data using SQL
query = """
SELECT DISTINCT name, email, status,
                purchase_amount
FROM customers
WHERE ~~customers~~ status = 'active';
"""

df = pd.read_sql(query, conn)
```

```
#step3: Export to Excel
df.to_excel("cleaned_customers_mysql.xlsx",
            index=False)
print("cleaned customer data exported to
      'cleaned_customers_mysql.xlsx'.")
conn.close()
```

SQLite Version

① No Installation Needed

SQLite comes with Python (sqlite3 module is built-in)

② Install Required Python Libraries:

pip install pandas openpyxl

③ Create Database and Table in SQLite

create a file called setup_sqlite.py and run this code:

```
import sqlite3

conn = sqlite3.connect("customerdb.sqlite")
cur = conn.cursor()

# Create table
cur.execute("""
CREATE TABLE IF NOT EXISTS customers(
id INTEGER PRIMARY KEY AUTOINCREMENT,
name TEXT,
email TEXT,
status TEXT,
purchase_amount FLOAT
)
""")
```

```
# insert data
cur.executemany("""
INSERT INTO customers (name, email, status, purchase_
                                amount) VALUES (?,?,?,?)
""", [
('Alice', 'alice@example.com', 'active', 250.50),
('Bob', 'Bob@example.com', 'inactive', 0),
('Alice', 'alice@example.com', 'active', 250.50),
('Eve', 'eve@example.com', 'active', 120.75),
('charlie', 'charlie@example.com', 'active', 300.00)
])
conn.commit()
conn.close()
```

④ Python script (SQLite) - wrangle_sqlite.py

```
import sqlite3
import pandas as pd

#step1: connect to sqlite
conn = sqlite3.connect("customerdb.sqlite")

#step2: clean data using SQL
query = """
SELECT DISTINCT name, email, status,
                purchase_amount
FROM customers
WHERE status='active';
"""

df = pd.read_sql(query, conn)

# Step3: Export to Excel
df.to_excel("cleaned_customers_sqlite.xlsx",
            index=False)
print("cleaned customerdata exported to
      'cleaned_customers_sqlite.xlsx'")
conn.close()
```

NoSQL Databases: MongoDB & Firebase in Datawrangling:

Non-national (NoSQL) data bases store data in formats like documents (JSON) or key-value pairs, offering flexible schema idea for modern apps. Popular NoSQL systems include

-> MongoDB - Document-based database storing JSON-like structures

-> Firebase (Firestore) - A cloud-based NoSQL database by Google, optimized for real time data.

Why use NoSQL for Data Wrangling:

* Schema flexibility
* Easy integration with apps
* Greate for Semi-structured data (eg. user profiles, sensor logs etc)

Setup and Installation:

MongoDB: (Local or Atlas cloud)

1. Install MongoDB:

   https://www.mongodb.com/try/download/community

2. Install Required libraries

   pip install pymongo pandas openpyxl

3. Start MongoDB locally or create a free cluster on

   https://www.mongodb.com/cloud/atlas.

Python script: Insert Unstructured Data into MongoDB

"insert_unstructured_mongo.py"

```
from pymongo import MongoClient
import pandas as pd

# Step1: connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
                                    # or Atlas URI
db = client["customerdb"]
collection = db["customers"]
```

#insert unstructured customer data

```
collection.insert_many([
{"name": "Alice", "email": "alice@example.com", "status": "active",
                    "purchase_amount": 250.50},
{"name": "Bob", "status": "inactive", "notes":
        "missing email and purchase"}, # missing fields
    name: Eve    email: "  status: active
    name: Zoe    email: "  status: "
    name: sam    email: "  status: "
                    "tags": ["loyal", "frequent"],
                    "purchase_amount": 120.75},
    "address": {"city": "NY", "zip": "10001"}},
    "purchase_history": [100, 200, 150]}
])

print("Unstructured data inserted into MongoDB")
```

Connect, filter and Export in python

dw_filter_export.py

```
from pymongo import MongoClient
import pandas as pd
from pandas import json_normalize

# step1: Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["customerdb"]
collection = db["customers"]

#step2: Retrive all documents
docs = list(collection.find())

#step3: Normalize (flatten) the documents
df = json_normalize(docs)

#step4: filter rows
#keep only active users with a valid purchase_amount
df_filtered = df[df["status"] == "active") &
              (df["purchase_amount"].notnull())]
```

```
# clean up columns
# Step5: Export to Excel
df_filtered.to_excel("filtered_customers.xlsx",
                     index=False)
print("Filtered data exported to 'filtered_customers.
      xlsx")
```

output

name	email	status	purchase_amount	tags
Alice	alice@ex.com	active	250.50	N/A
Eve	eve@ex.com	active	120.75	['loyal', 'frequent']

Firebase (cloud store)

1. Goto firebase Console
2. Create a new project → Add firestore database
3. Download your service account JSON:
   * project settings → Service Accounts → Generate new private key

4. Install python Libraries:

   pip install firebase-admin pandas openpyxl

Python Connection & Export Example

* dw_firebase_to_excel.py

```
import firebase_admin
from firebase_admin import credentials, firestore
import pandas as pd

# Step1: Initialize firebase
cred = credentials.Certificate("path/to/your/firebase-key.json") # Replace
firebase_admin.initialize_app(cred)
db = firestore.client()
```

```
# step 2: Read data from firestore
docs = db.collection("customers").stream()
raw_data = []
for doc in docs:
    data = doc.to_dict()
    raw_data.append(data)
df = pd.DataFrame(raw_data)
# step3: filter and deduplicate
df = df[df['status'] == 'active']
df = df.drop_duplicates(subset=["name",
                                "email"])
# step4: Export to Excel
df.to_excel("cleaned_customers_firebase.xlsx",
            index=False)
print("firebase cleaned data exported to
      'cleaned_customers_firebase.xlsx")
```

Step 1: Setup Firebase project

1. Go to https://console.firebase.google.com/
2. Click "Add project" → follow the prompts
3. Enable Firestore Database:
   - In the left menu: Build ⇒ Firestore Database
   - Choose "Start in test mode"
   - Click Create database.

Step 2: Generate Firebase Admin SDK Key

1. Go to project Settings → Service Accounts
2. Account clicks "Generate New Private Key"
3. This downloads a .json file ⇒ Save it (eg: firebase-key.
4. This file will be used in your python scri to connect.

Create and Insert Sample Data

sample_data.py

```
import firebase_admin
from firebase_admin import credentials,
                           firestore

# step1: Initialize firebase
cred = credentials.Certificate("firebase-key.json")
                    # Replace with your path
firebase_admin.initialize_app(cred)
db = firestore.client()

Step2: Create unstructured customer data
customers = [
  {"name": "Alice", 'email':   status: "active"
                          purchase_amount : 250}
  ,,
  ,,
  ,,
]
```

step 3: Insert into firestore

```
for customer in customers:
    db.collection('customers').add(customer)
print("Sample customer data added to Firebase Firestore")
```