**Automated Data Extraction, Cleaning, and Tabular Display from URLs**

**Objective:**

To develop a **Python-based automated system using Jupyter Notebook** that accepts a user-provided **data source URL** (PDF, CSV, Excel, JSON, HTML, .data, .txt), detects the format, extracts the data, performs cleaning operations, and displays it in a well-structured tabular format.

**Problem Statement:**

Data analysts and data scientists often encounter unstructured or differently formatted data hosted online. These datasets can be:

* In various formats (CSV, Excel, PDF, HTML, JSON, plain text)
* Poorly structured or dirty (missing values, extra spaces, bad formatting)
* Without headers or inconsistent delimiter usage

There is no single tool to handle **automated detection, parsing, and cleaning** of such varied sources.

**Proposed Solution:**

Create a Jupyter-based interactive tool that:

1. **Takes a URL input**
2. **Detects the data format (content-type + file extension)S**
3. **Parses the content accordingly**
4. **Cleans the data using common data wrangling rules**
5. **Displays a preview**
6. **Exports the cleaned output to output\_table.csv**

**Project flow:**

**📥 User Input (URL)**

**↓**

**🔍 Detect File Type (based on URL or fallback)**

**↓**

**📂 Download / Extract Data**

**↓**

**🧹 Clean / Transform Data**

**↓**

**📋 Display as Table (Text + HTML)**

**Key Features:**

| **Feature** | **Description** |
| --- | --- |
| **Universal Format Support** | Detect and parse CSV, JSON, PDF, Excel, HTML, .data, and plain text files |
| **Smart Format Detection** | Uses content-type headers + file extension + known patterns |
| **Data Cleaning Pipeline** | Removes nulls, trims strings, parses dates, fills missing values |
| **Interactive Tabular Output** | Shows preview using tabulate and IPython.display |
| **Output Saving** | Saves cleaned data as output\_table.csv |
| **UCI .data file compatibility** | Parses .data and .txt from UCI ML repositories |
| **Plain text delimiter handling** | Tries common delimiters like comma, tab, pipe, etc. |

**Technology Stack:**

* **Language:** Python 3.x
* **Environment:** Jupyter Notebook
* **Libraries Used:**
  + pandas – Data manipulation
  + requests – URL content fetching
  + pdfplumber – PDF text extraction
  + BeautifulSoup – HTML parsing
  + openpyxl – Excel parsing
  + tabulate, IPython.display – Output formatting

**Data Cleaning Steps:**

1. Strip column names
2. Remove fully empty rows/columns
3. Drop duplicates
4. Fill numeric missing values with median
5. Fill categorical missing values with mode
6. Strip whitespace from strings
7. Parse columns containing "date" into datetime
8. Remove inconsistent encoding if any

**Data Extraction:**

* **CSV & JSON**: Reads directly into pandas DataFrames.
* **PDF**: Extracts text using pdfplumber.
* **Excel**: Processes .xls and .xlsx files with openpyxl.
* **HTML**: Parses tables using BeautifulSoup and pandas.read\_html.
* **Plain Text**: Attempts to parse using common delimiters (comma, tab, semicolon, pipe, space).

**Code & Explain:**

1. **Install Required Libraries:**

!pip install pandas tabulate requests PyPDF2 pdfplumber beautifulsoup4 lxml

!pip install html5lib

!{sys.executable} -m pip install html5lib

!pip install openpyxl

| **Library** | **Purpose** | **Required for** |
| --- | --- | --- |
| pandas | Data manipulation | Core for CSV, JSON, Excel, HTML, tables |
| tabulate | Console table output | Pretty print tables |
| requests | Download files | PDF, JSON, HTML from URLs |
| PyPDF2 | Basic PDF handling | Optional, not ideal for table extraction |
| pdfplumber | Accurate text/table from PDFs | Best PDF parser for structured content |
| beautifulsoup4 | HTML parsing | HTML scraping with custom logic |
| lxml | Fast XML/HTML parser | Needed by pandas.read\_html, bs4 |
| html5lib | Robust HTML parser | Needed when pandas.read\_html(flavor='html5lib') |
| openpyxl | Excel reader engine for .xlsx files | Required for Excel file support |

1. **Import Libraries:**

import sys

import pandas as pd

import requests

import json

import pdfplumber

from bs4 import BeautifulSoup

from io import BytesIO

from tabulate import tabulate

from IPython.display import display, HTML

1. **Get Link Input From User:**

url = input("Enter the link (PDF, CSV, PDF, XLS, JSON, HTML): " # Take user input

1. **Detect file type:**

def detect\_file\_type(url):

try:

head = requests.head(url, allow\_redirects=True, timeout=10)

content\_type = head.headers.get("Content-Type", "").lower()

#create function to detect\_file

#request.head get only header not full content .

# allow\_redirects=True: Follows any URL redirections.

# timeout=10: Waits up to 10 seconds for a response before giving up

if "text/csv" in content\_type:

return "csv"

elif "application/json" in content\_type:

return "json"

elif "application/pdf" in content\_type:

return "pdf"

elif "excel" in content\_type or "spreadsheet" in content\_type:

return "excel"

elif "text/plain" in content\_type:

return "plain\_text"

elif "html" in content\_type:

return "html"

except:

pass

# It checks the file type from the URL’s response header (Content-Type) and returns the type as 'csv', 'json', 'pdf', 'excel', 'plain\_text', or 'html' based on the content.

1. **Fallback: file extension or known patterns:**

url = url.lower()

if url.endswith(".csv"):

return "csv"

elif url.endswith(".json"):

return "json"

elif url.endswith(".pdf"):

return "pdf"

elif url.endswith(".xls") or url.endswith(".xlsx") or "1drv.ms" in url:

return "excel"

elif url.endswith(".data") or url.endswith(".txt") or "archive.ics.uci.edu" in url:

return "plain\_text"

elif url.endswith(".html"):

return "html"

else:

return "unknown"

file\_type = detect\_file\_type(url)

print("Detected file type:", file\_type)

#first url converted into lower case and checked type of url by (url.endswith)

#if doesn’t match retun unknown

#call the function with URL and URL type & Display result

1. **Extract Data:**

def extract\_data(url, file\_type):

#define function extract data that take two input

#url: the link to the data file.

#file\_type: the detected type of the file (like csv, json, pdf, etc.).

try:

# try the following code, and if anything goes wrong, handle the error.

if file\_type == "csv":

return pd.read\_csv(url)

# If the file is a CSV, read it directly using pandas and return the table (DataFrame).

elif file\_type == "json":

response = requests.get(url)

data = response.json()

return pd.json\_normalize(data)

#fetch data using request convert json to dictionary normalize and written as dataframe

elif file\_type == "pdf":

response = requests.get(url)

with pdfplumber.open(BytesIO(response.content)) as pdf:

text = ''.join(page.extract\_text() + '\n' for page in pdf.pages)

return pd.DataFrame([{"PDF\_Content": text.strip()}])

#request used to pdf download, pdfplumber to open it, extract pages store in table

elif file\_type == "excel":

response = requests.get(url)

return pd.read\_excel(BytesIO(response.content), engine='openpyxl')

#Download the Excel file, Read it using pandas and return the content as a table.

elif file\_type == "html":

response = requests.get(url)

tables = pd.read\_html(StringIO(response.text))

return tables[0] if tables else pd.DataFrame([{"Message": "No tables found in HTML"}])

#try to find “html” string in url

#If it doesn’t match but the URL is still a webpage (http...), it **assumes it’s HTML**

#Download the webpag & Try to extract any HTML tables.

#stringIO(respose.text) wrap string into file like object for read by pandas

#If a table is found, return the first one.

#If none found, return a table that says “No tables found”.

elif file\_type == "plain\_text":

response = requests.get(url)

content = response.content.decode('utf-8')

return try\_parse\_plain\_text(content)

#Download the text file & Decode it from bytes to string (UTF-8).

#Try to convert it into a table using your helper function try\_parse\_plain\_text().

else:

return pd.DataFrame([{"Error": "Unsupported file type or could not detect properly."}])

#If the file type isn't supported, return a DataFrame with an error message.

except Exception as e:

return pd.DataFrame([{"Error": str(e)}])

#If **any error occurs** in the above code (like a network error or parsing issue), catch it and return a table with the error message.

1. **Try parsing plain text with delimiters:**

def try\_parse\_plain\_text(text):

for delimiter in [',', '\t', ';', '|', ' ']:

try:

df = pd.read\_csv(StringIO(text), delimiter=delimiter, engine='python', header=None)

if df.shape[1] > 1:

print(f"✅ Parsed plain text using delimiter: '{delimiter}'")

return df

except:

continue

return pd.DataFrame([{"Error": "❌ Could not parse plain text with common delimiters"}])

‘’’

Define function

take input text

function will try **each one** to see which best separates the data into columns.

inside the loop, we **try to read the text as a CSV** using the current delimiter:

StringIO(text) turns the plain text string into a file-like object.

delimiter=... tells pandas how to split the columns.

engine='python' allows flexible parsing.

header=None treats the first row as normal data (no column names).

This checks how many columns were created.

df.shape[1] is the **number of columns**.

If there is **more than one column**, the delimiter likely worked properly.

Print if successful print which delimiter worked & return DF

If parsing throws an error (invalid format, bad delimiter, etc.), it **skips** that delimiter and **tries the next one**.

If **none** of the delimiters work, it returns a DataFrame with an error message.

‘’’

1. **Clean the DataFrame:**

def clean\_data(df):

print("\n🔍 Initial Data Shape:", df.shape)

df.columns = df.columns.astype(str).str.strip() # convert to string and remove spaces

df = df.dropna(axis=1, how='all') #drops empty coloumn

df = df.dropna(axis=0, how='all') #drops empty rows

df = df.drop\_duplicates() #Ensures no duplicate entries

#for interger

for col in df.select\_dtypes(include='number').columns:

df[col] = df[col].fillna(df[col].median())

# find all numeric, float values and missing values replace by medium of coloum

#for text objects

for col in df.select\_dtypes(include='object').columns:

if df[col].isnull().any():

mode = df[col].mode()

if not mode.empty:

df[col] = df[col].fillna(mode[0])

df[col] = df[col].astype(str).str.strip()

#find text missing values replace with most frequent values

#look for date time coloums

for col in df.columns:

if 'date' in col.lower():

try:

df[col] = pd.to\_datetime(df[col], errors='coerce')

except:

pass

# Convert in proper date format if fails shows NaT

print("✅ Cleaned Data Shape:", df.shape)

return df

# Shows the final shape after cleaning & Returns the cleaned DataFrame

1. **Run full pipeline:**

df = extract\_data(url, file\_type) #call extract dat function

if 'Error' in df.columns or df.empty: #check for error

print("\n⚠️ Error or Empty Data:")

print(tabulate(df, headers='keys', tablefmt='pretty')) #Tabulate print DF in table format

else:

df = clean\_data(df) #lean\_data() function remove duplicates, fills NaNs, strips text etc)

print("\n📋 Table Preview:")

print(tabulate(df.head(10), headers='keys', tablefmt='pretty'))

#Displays the **first 10 rows** of the cleaned table in the console using tabulate().

print("\n🔍 HTML Preview:")

display(HTML(df.head(10).to\_html(index=False)))

#Uses display() and to\_html() to show the same first 10 rows in a html table directly inside the Jupyter Notebook.

df.to\_csv("output\_table.csv", index=False)

print("\n💾 Saved as output\_table.csv")

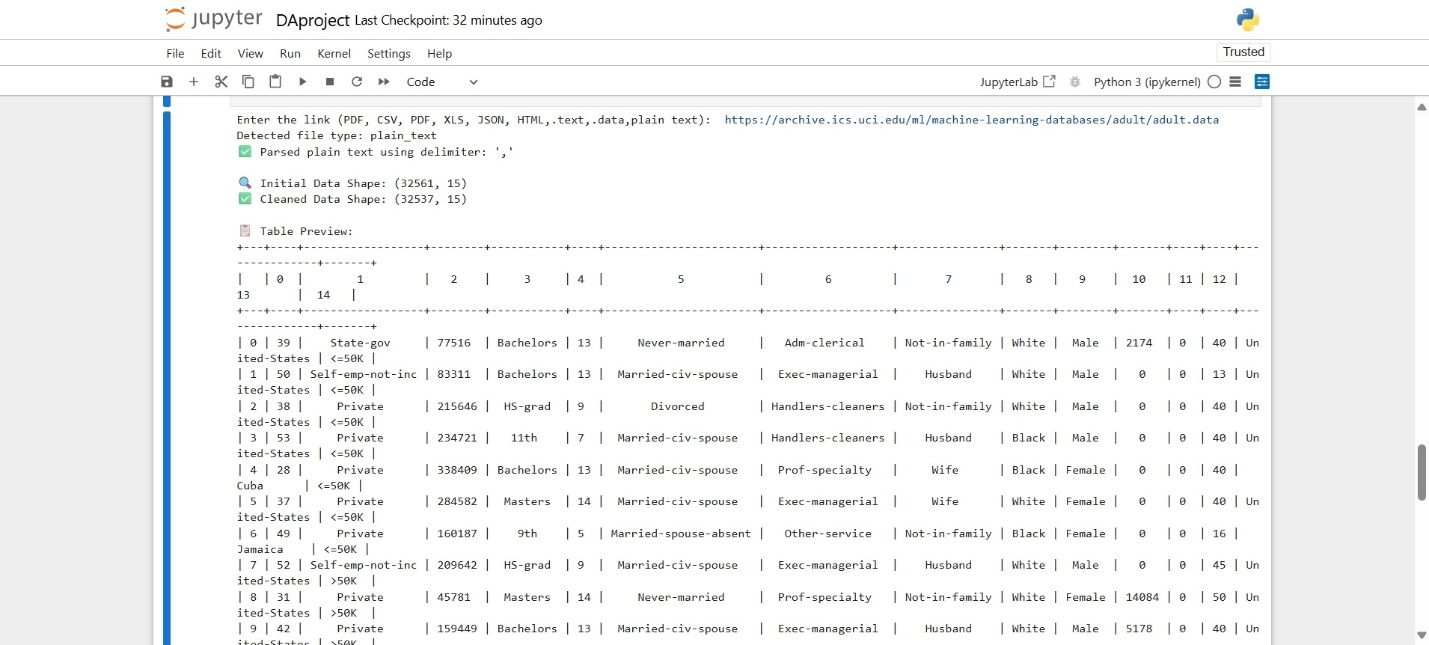
#cleaned msg show confirm save with msg

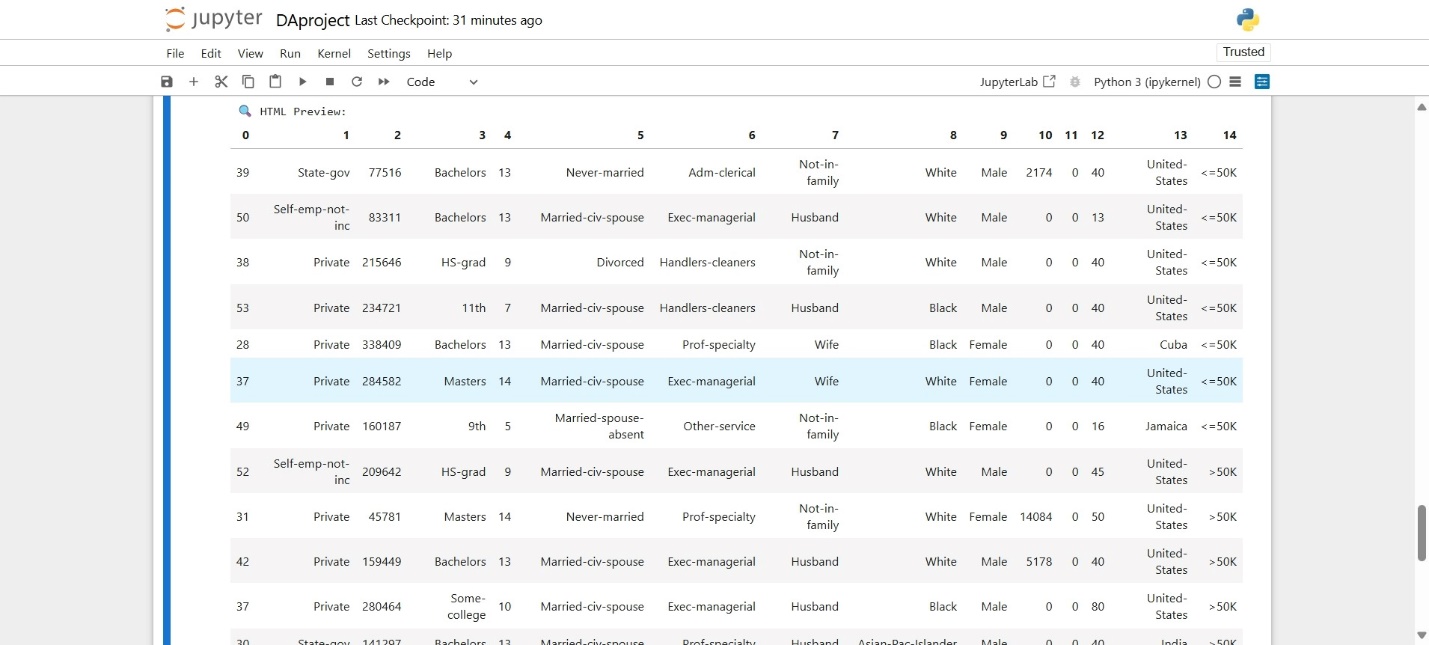
**Deliverables:**

* Final Jupyter Notebook (.ipynb)
* Sample test links and outputs
* output\_table.csv (cleaned data export)
* Optionally, a Python script version for CLI use

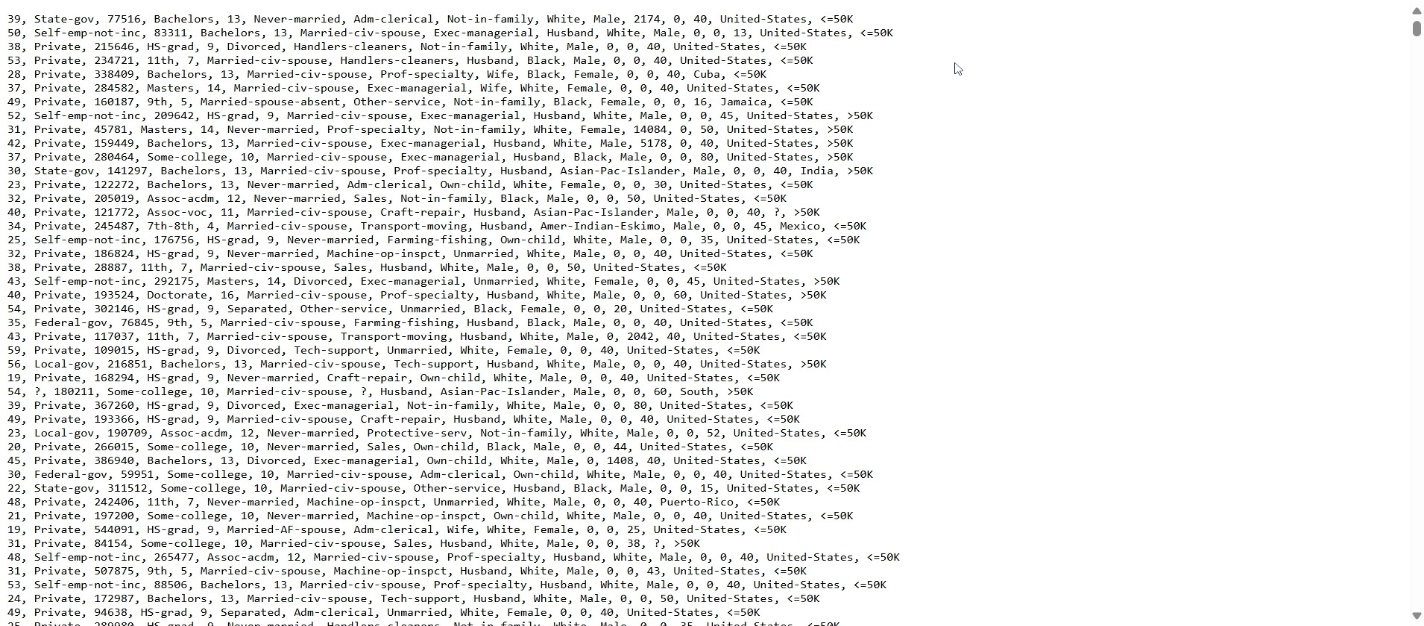
**Output1:**



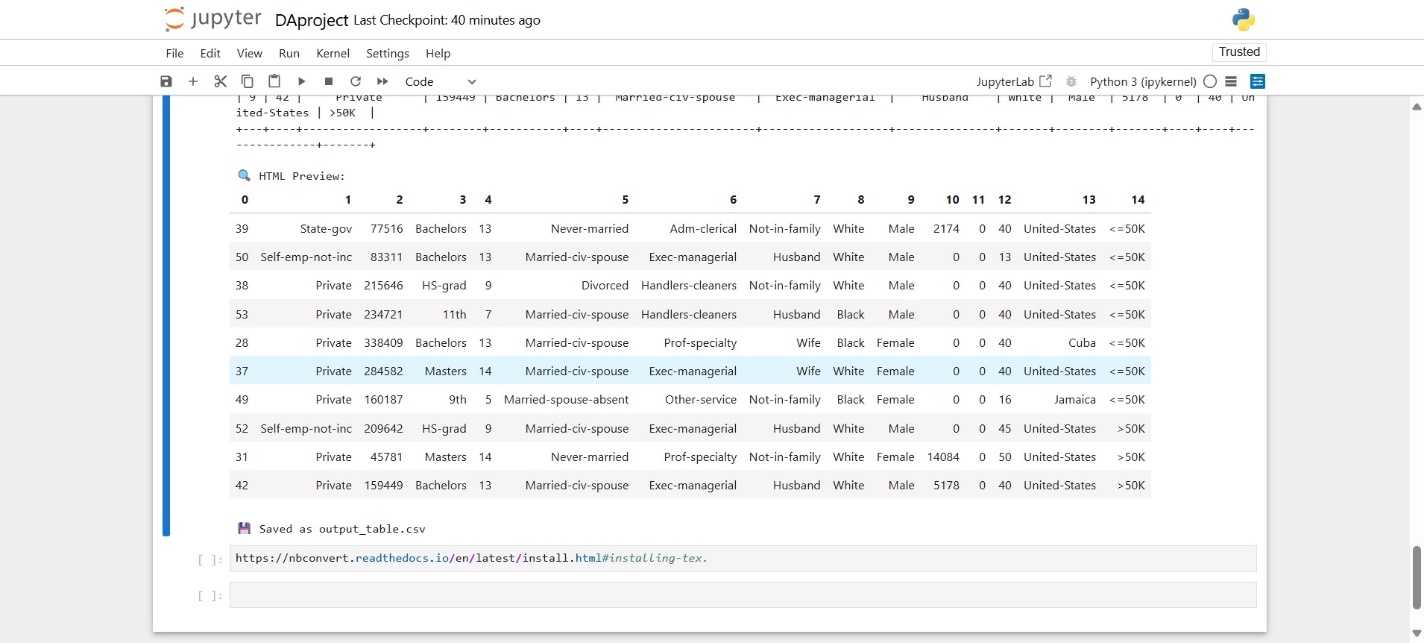




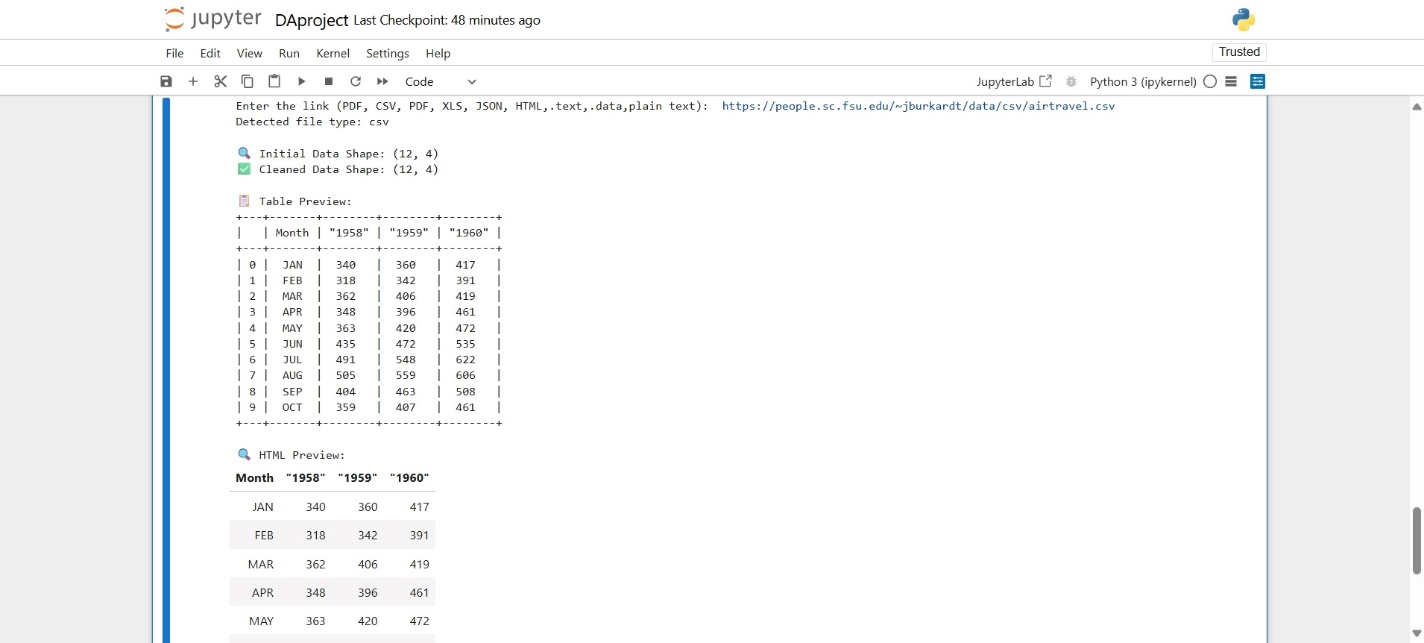
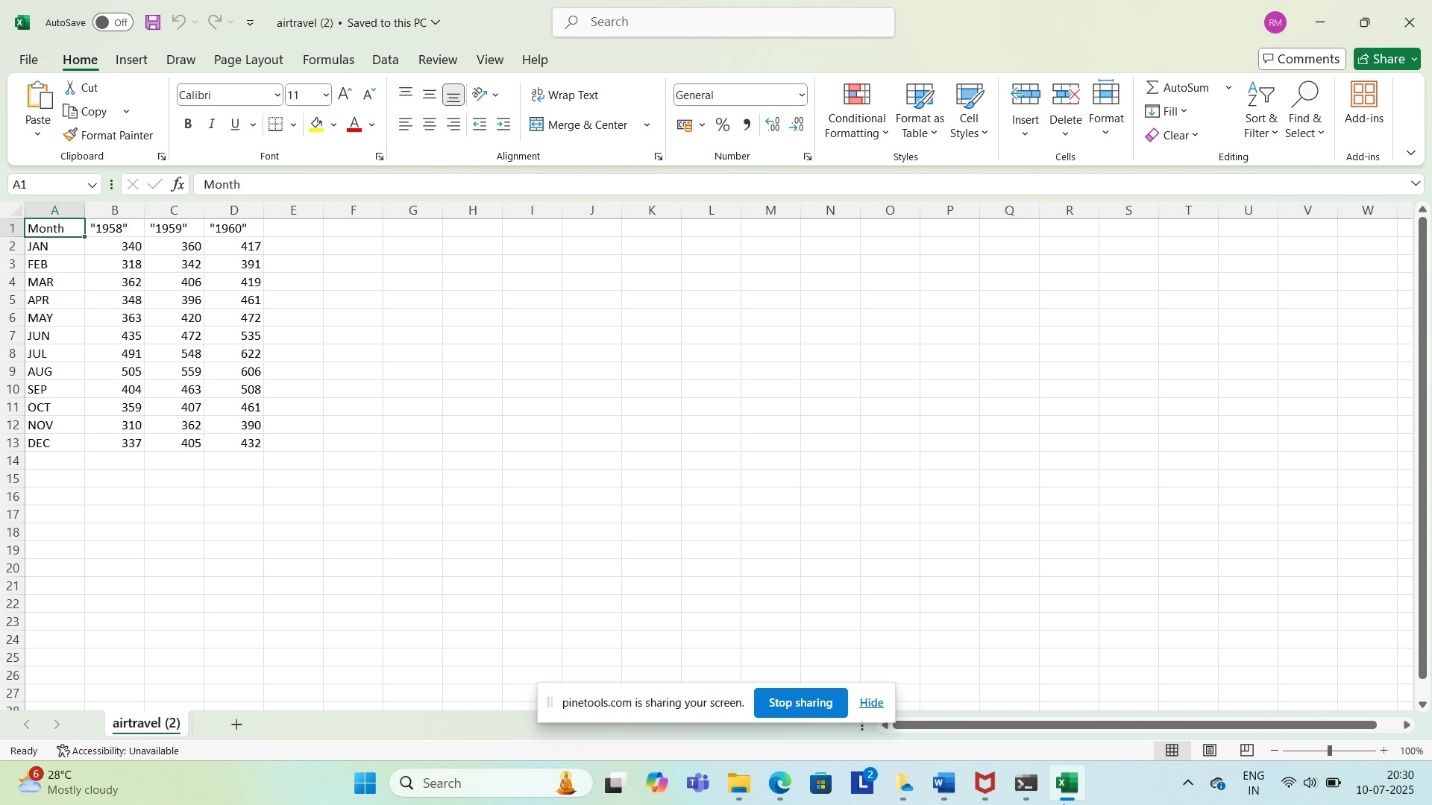
**Output 2:**







**Output3:**



**Conclusion:**

This tool significantly reduces the effort required to extract and clean datasets from diverse sources. It empowers data analysts to get to the **insights stage faster**, with a single-click, format-agnostic ingestion and cleaning pipeline.

**Sample Links:**

<https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf>

https://unec.edu.az/application/uploads/2014/12/pdf-sample.pdf

<https://people.sc.fsu.edu/~jburkardt/data/csv/airtravel.csv>

<https://raw.githubusercontent.com/selva86/datasets/master/BostonHousing.csv>

<https://randomuser.me/api/?results=10>

<https://raw.githubusercontent.com/typicode/demo/master/db.json>

<https://www.worldometers.info/world-population/population-by-country/>

<https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)>

<https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data>

<https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>

[archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data](https://archive.ics.uci.edu/ml/machine-learning-databases/auto-mpg/auto-mpg.data)

**Reference Links:**

[Automated Data Extraction and Transformation Using Python, OpenAI, and AWS | Data Science Blog](https://nycdatascience.com/blog/python/automated-data-extraction-and-transformation-using-python-openai-and-aws/)