AIG150- Week 2

Working With Real Data

Reading Text:

Chap 04, 06: Pandas for everyone

Chap 06: Python for Data Science for Dummies

Agenda

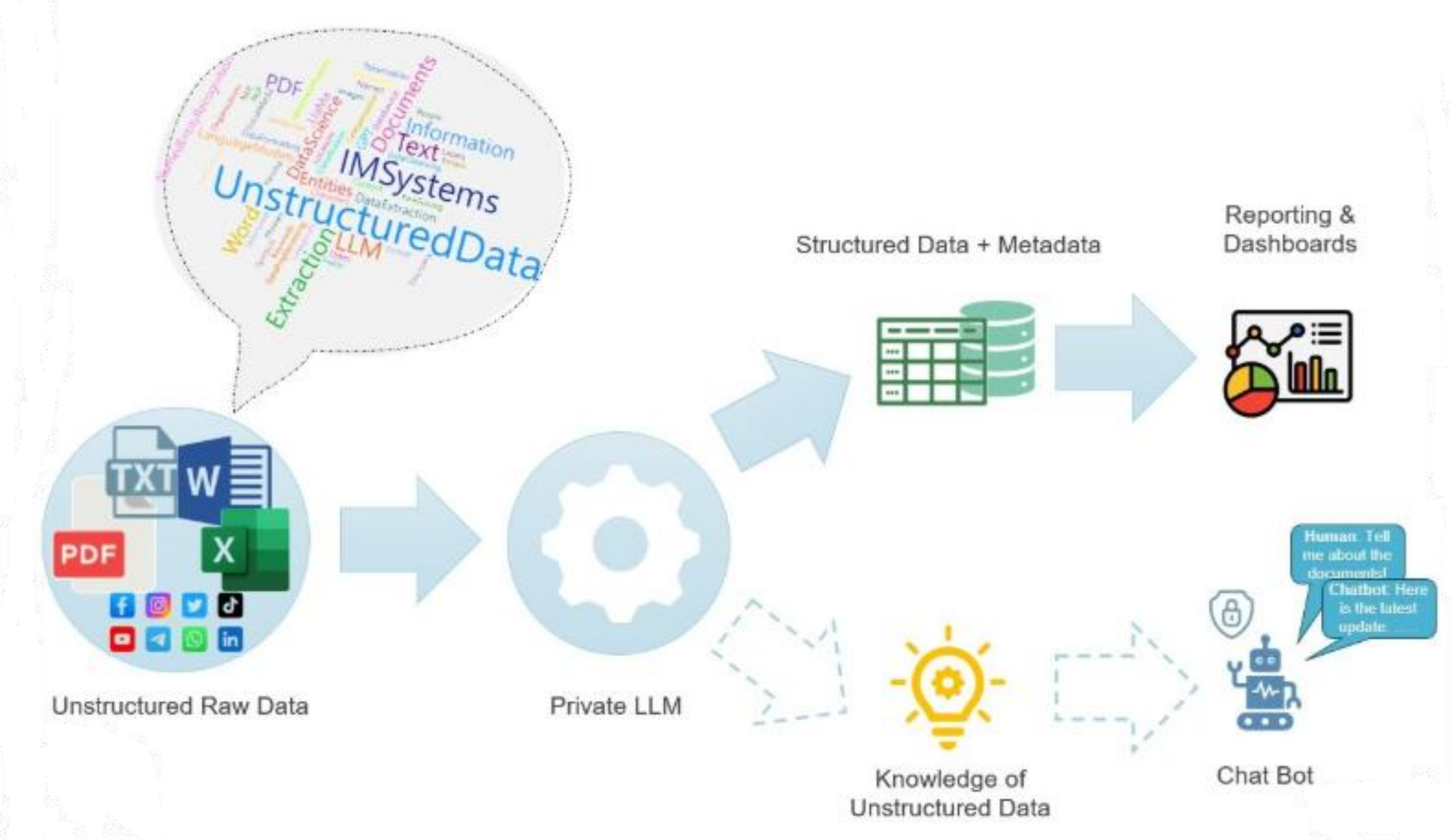
Data Preparation Topics

- Importing and Exporting Data
- Streaming and Sampling Data
- Accessing Structured Flat-File Data
- Handling Unstructured File Data
- Managing Data from Relational Databases
- Accessing Data from the Web

Data Transformation & Cleaning

- Data Cleaning
- Data Assembly
- Tidy Data Principles
- Concatenation
- Merging Multiple Datasets

Multiple Types of Data



Streaming, and Sampling Data

- The most efficient way to work with data is to load it directly into memory.
- For small files, you can read the entire file at once using file.read(). (Not suitable for very large files.)
- Streaming: Process the data in smaller chunks or download it piece by piece to avoid delays.
- Sampling: Retrieve only a subset of records instead of the entire dataset.

Accessing Data in Structured Flat-File Form

- Text file
- Csv file
- Excel file

Sending Data in Unstructured File Form

- Unlike CSV or Excel files that store data in rows and columns, these files have no fixed structure.
- They consist of a sequence of bits and require an interpretation algorithm to extract meaningful information.
- The file header usually provides hints about the type of data stored.
- Common examples include image, audio, and video files.

Managing Data from Databases

- Relational databases include SQL, PostgreSQL, Oracle, and others.
- To read data from a table, you can use the read_sql_table() method. The create_engine() function is used to establish a connection using a database URI.
- NoSQL databases, such as MongoDB, can be accessed using PyMongo.

Accessing Data from the Web

- Data collected from web services and microservices
- XML and JSON

What is Tidy Data ?

- Each row is an observation
- Each column is a variable
- Each type of observational unit forms a table

Which One Is Tidy ????

Table 1

Country	Year	Cases	Population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898

Table 2

Country	Year	type	count
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	745
Brazil	1999	cases	37737
Brazil	2000	population	174504898

Same data spread across two tables

Country	1999	2000
Afghanistan	745	266
Brazil	37737	80488

Country	1999	2000
Afghanistan	19987071	20595360
Brazil	172006362	174504898

Merging Multiple Datasets

Identify:

- What needs to be combined ?
- Do we need to concatenate or join the data ?
- The appropriate function for merging data sets
- Assess if the merge was proper

Data Preparation with Python: From Raw to Ready

- Know where data lives and how to get it.
- Apply repeatable patterns: import → clean → assemble → tidy → export.
- Work with large files and multiple formats.
- Produce analysis-ready tables.

1) Importing & exporting data

Scenario: marketing exports a big orders_2025-09.csv from Shopify; finance wants a clean parquet file for analytics.

Steps

Inspect a few rows; 2) enforce types/dates; 3) select/rename only what you need; 4) save in an efficient format.

```
import pandas as pd orders =
pd.read_csv("orders.csv",
        parse_dates=["created_at"],
        dtype={"order_id":"int64","customer_id":
        "int64","currency":"category"})
orders.dtypes
orders.to_csv("orders_copy.csv", index=False)
# Parquet if pyarrow installed #
orders.to_parquet("orders.parquet", index=False)
```

- Date,Product,Revenue
- 2025-01-01,Shirt,120
- Date object # still a string
- Date datetime64[ns] # converted to datetime

- Parquet is a columnar storage format (used by big data systems like Spark, Hive, AWS Athena).
- It is more efficient than CSV because:It stores columns together instead of rows.
- It supports compression (smaller file sizes).
- It allows faster reading of large datasets.

Streaming & Sampling

- Stream orders_large.csv by chunksize.
  - Instead of loading the whole file into memory at once, pandas reads the file in **smaller pieces (chunks)**.
- Compute a running metric. notna() is a pandas **function to detect non-missing values**.
- Take a small random sample.

```
import pandas as pd
total = 0.0;
count = 0
for chunk in pd.read_csv("orders_large.csv", chunksize=500):
      total += chunk["total_price"].sum()
      count += chunk["total_price"].notna().sum()
print("Average order total:", total/count)
sample = pd.read_csv("orders_large.csv").sample(10, random_state=42)
sample.head()
```

Structured Flat Files

CSV/TSV: delimited, simple, universal.
Excel: business-friendly sheets; pick sheets & ranges.
Control **delimiter, encoding, NA tokens**.
Use cols, dtype, parse_dates, na_values.

```
import pandas as pd
cust = pd.read_csv("customers.csv",
        dtype={"customer_id":"int64"})
cust.to_csv("customers.tsv", sep="\t", index=False)
pd.read_csv("customers.tsv", sep="\t").head()
# Excel example (if you add an .xlsx file later)
# roles =pd.read_excel("roles.xlsx",sheet_name="role_map")
```

Unstructured Data

JSON: nested; flatten to tables.
Logs: free text; extract with regex.
show “record_path + meta”; emphasize schema choices.

```
import json, re, pandas as pd
from pandas import json_normalize
payload =
json.load(open("orders_nested.json"
))
flat = json_normalize(payload,
record_path=["orders","items"],
        meta=[["orders","order_id"],
["orders","channel"]])
flat.rename(columns={"orders.order_
id":"order_id","orders.channel":
                "channel"},
inplace=True)
```


```
{ "orders":
        [
{
 "order_id": 101, "channel":
"online",
"items": [
{"item_id": "A1", "price": 50},
{"item_id": "A2", "price": 30} ]
},
{
"order_id": 102, "channel":
"store",
"items": [
{"item_id": "B1", "price": 20} ]
}
] }
```

Unstructured Data

```
rows=[];
pat=re.compile(r'(?P<ip>\S+).*"(?P<verb>GET|POST) (?P<path>\S+)')
for line in open("access.log"):
        m=pat.search(line);
        if m: rows.append(m.groupdict())
pd.DataFrame(rows).head()
```

```
?P<name> → names the group, so it can be returned as a dictionary.
(?P<ip>\S+)              # capture first non-space string as 'ip'
.*                       # skip the middle part
"(?P<verb>GET|POST)      # capture HTTP verb (GET or POST) as 'verb'
 (?P<path>\S+)           # capture requested path (non-space) as 'path'
```

```
192.168.1.1 - - [15/Sep/2025] "GET /index.html HTTP/1.1" 200 2326
```

Matches:

- ip → "192.168.1.1"
- verb → "GET"
- path → "/index.html"

Relational Databases

- SQLite .db: serverless, perfect for class.
- Use SQL to select, Pandas to analyze.
    - Connect to school.db.
    - Query into DataFrame.
    - Close connection.

```python
import sqlite3, pandas as pd
con = sqlite3.connect("school.db")
students = pd.read_sql("SELECT * FROM students", con)
con.close()
students.head()
```

Accessing Data from the Web

- APIs & HTML Tables
  - requests for JSON APIs.
  - pd.read_html for simple tables.

```
import pandas as pd
tables = pd.read_html("stats.html")
tables[0]
# API example if online:
# import requests, pandas as pd
# r = requests.get("https://api.exchangerate.host/latest",
      timeout=15);
r.raise_for_status()
# rates = pd.DataFrame(list(r.json()["rates"].items()),
      columns=["currency","rate"]) # rates.head()
```

Data Cleaning

- Types & dates
- Missing values
- String cleanup
- Deduplication
  - if deduping, pick a **business key** (email/customer_id) and rule (keep latest).

```
import pandas as pd
cust = pd.read_csv("customers.csv")
cust.columns = (cust.columns.str.strip().str.lower()
                .str.replace(r"[^0-9a-zA-Z]+","_", regex=True))
cust["email"] =cust["email"].astype("string").str.strip().str.lower()
cust["country"] = cust["country"].fillna("Unknown")
cust.head()
```

`.str.strip()`

- Removes leading and trailing spaces. ['Customer Name', 'Order-ID', 'Order Amount($)']
- .str.replace ['customer_name', 'order_id', 'order_amount_']

Data Assembly

- Join tables (orders ↔ customers).
- Feature engineering (aggregations).
    - call out grain (row definition) before aggregating.

```python
import pandas as pd
orders = pd.read_csv("orders.csv")
customers = pd.read_csv("customers.csv")
wide = pd.merge(orders, customers,
        on="customer_id", how="left", validate="many_to_one")
```

Data Assembly

order_id	customer_id	amount
1	101	250
2	102	300
3	101	150

order_id	customer_id	amount	name
1	101	250	Alice
2	102	300	Bob
3	101	150	Alice

customer_id	name
101	Alice
102	Bob
103	Charlie

`validate="many_to_one"`

- Ensures merge relationship:
  - `orders` → **many rows per customer** (many orders).
  - `customers` → **only one row per customer** (unique customers).
- If `customers` has duplicates in `customer_id`, pandas will raise an error.

Data Assembly

```
import pandas as pd
aov = (orders.groupby("customer_id")["total_price"]
        .mean().rename("avg_order_value"))
```

order_id	customer_id	total_price
1	101	250
2	102	300
3	101	150
4	103	400

Grouping by `customer_id` → 3 groups:

- Customer 101 → [250, 150]
- Customer 102 → [300]
- Customer 103 → [400]

```
customer_id
101    200.0   # (250 + 150) / 2
102    300.0
103    400.0
```

```
customer_id
101    200.0
102    300.0
103    400.0
Name: avg_order_value, dtype: float64
```

Data Assembly

```python
import pandas as pd
model_df = wide.merge(aov, on="customer_id", how="left")
model_df.head()
```

Join wide with aov on customer_id.how="left" → keep all rows from wide (left DataFrame).
If a customer doesn't have an aov, it will show NaN.

customer_id	name	region	avg_order_value	
101	Alice	East	200.0	
102	Bob	West	300.0	
103	Charlie	North	400.0	
104	Diana	South	NaN	# no orders, so no AOV

How To Tidy Data Using Python

- **melt()** unpivots a DataFrame from wide to long format
- **Pivot()** reshapes DataFrame organized by given index/ column values
- See the sample code provided with the lecture

pandas.DataFrame.melt

- melt() reshapes your **wide data** into **long (tidy) format**.

- Wide format = one row per student, many columns for subjects.
- Melted (long) format = one row per (student, subject) pair, with the score as a value.
- id_vars = columns that identify who or what the data is about.
- var_name = tells you which variable this row refers to.
- value_name = holds the actual measured data.

wide data into long (tidy) format

Wide Data

```
import pandas as pd df =
pd.DataFrame({ "student":
["Alice", "Bob"], "math": [90,
85], "science": [80, 95],
"english": [88, 92] })
print(df)
```

	student	math	science	english
0	Alice	90	80	88
1	Bob	85	95	92

Melted (long) Data

```
long_df = df.melt(
id_vars=["student"], # keep student
as identifier var_name="subject", #
new column name for old headers
value_name="score" # new column
name for values )

print(long_df)
```

	student	subject	score
0	Alice	math	90
1	Bob	math	85
2	Alice	science	80
3	Bob	science	95
4	Alice	english	88
5	Bob	english	92

Pivot Back to Wide

```
wide_again = long_df.pivot(
index="student", # becomes the row index
columns="subject", # becomes the new wide
columns values="score" # fills the table
with these values )

print(wide_again)
```

Subject/ Student	english	math	science
Alice	88	90	80
Bob	92	85	95

Pivot Table with Aggregation

- melt() → long format (good for tidy analysis, plotting, groupby).
- pivot() → back to wide format (one row = one entity, many columns).
- pivot_table() → same, but can handle duplicates using aggregation.

```
long_dup = pd.DataFrame({ "student":
["Alice","Alice","Bob","Bob"], "subject":
["math","math","science","science"],
"score": [90,95,80,85] })

pivoted = long_dup.pivot_table(
index="student", columns="subject",
values="score",
aggfunc="mean" # or sum, max, min, etc. )

 print(pivoted)
```

Subject/ Student	math	science
Alice	92.5	NaN
Bob	NaN	82.5

Concatenation

- Stack rows from same schema (e.g., weekly files).
- Or side-by-side with axis=1 when indices align.

```
import pandas as pd
w36 = pd.read_csv("sales_w36.csv")
w37 = pd.read_csv("sales_w37.csv")
all_sales = pd.concat([w36, w37], ignore_index=True)
all_sales.head()
```

Merging Multiple Datasets

- Bring entities together (orders + customers).
- Choose the join type (left/inner/right).
    - validate joins; watch out for one-to-many explosions.

```
import pandas as pd
orders = pd.read_csv("orders.csv")
customers = pd.read_csv("customers.csv")
orders_customers = pd.merge(orders, customers,
                             on="customer_id", how="left")
orders_customers.head()
```

Summary

- Pipeline recap:
  - Import → 2) Clean → 3) Assemble → 4) Tidy → 5) Export
    Deliverables: clean CSV/Parquet + notebook with steps & rationale.
- Import customers.csv, clean emails, fill missing country, export.
- Stream orders_large.csv, compute global mean total_price.
- Melt wide_sales.csv to (store, month, sales), then pivot back.
- Merge orders + customers, compute avg_order_value per customer.

- **Pattern:** read → check → fix types → handle NA → dedupe → join → tidy → export

- “One row per observation. One column per variable. One table per unit.”