Course Title:

Data Science (数据科学)

(Semester: Fall 2021)

Dr. Oluwarotimi W. SAMUEL

Research Center for Neural Engineering
Shenzhen Institutes of Advanced Technology
Chinese Academy of Sciences

Contact: (Email: samuel@siat.ac.cn & timitex92@gmail.com)

Phone: +86-15814491870

(2021.10.14)

Digital Data Description and Manipulation

❑ Outline for today's lecture

- ✓ Unstructured Data
- ✓ Structured Data
- ✓ Data Manipulation

Data Description & Manipulation

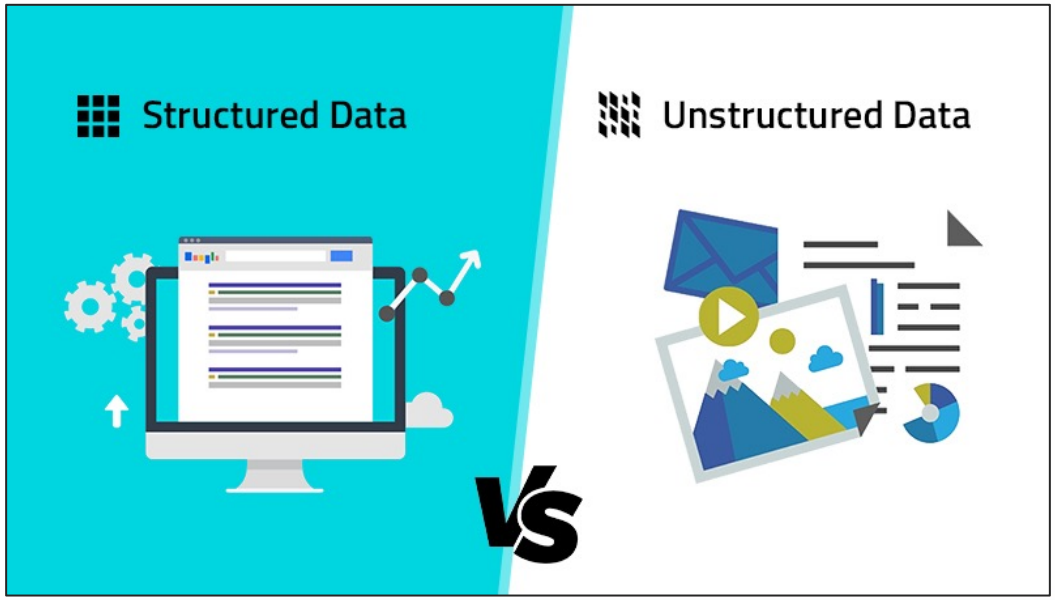
- **Objective:** This lecture will focus on discussing classification of digital data with a view of providing concise description and manipulation approaches.

- **Expectation:** At the end of this lecture, students are expected to understand the basic classification of digital data and how such data could be manipulated/handled.

Structured Data and Unstructured Data

❑ Data, the key resource in Data Science

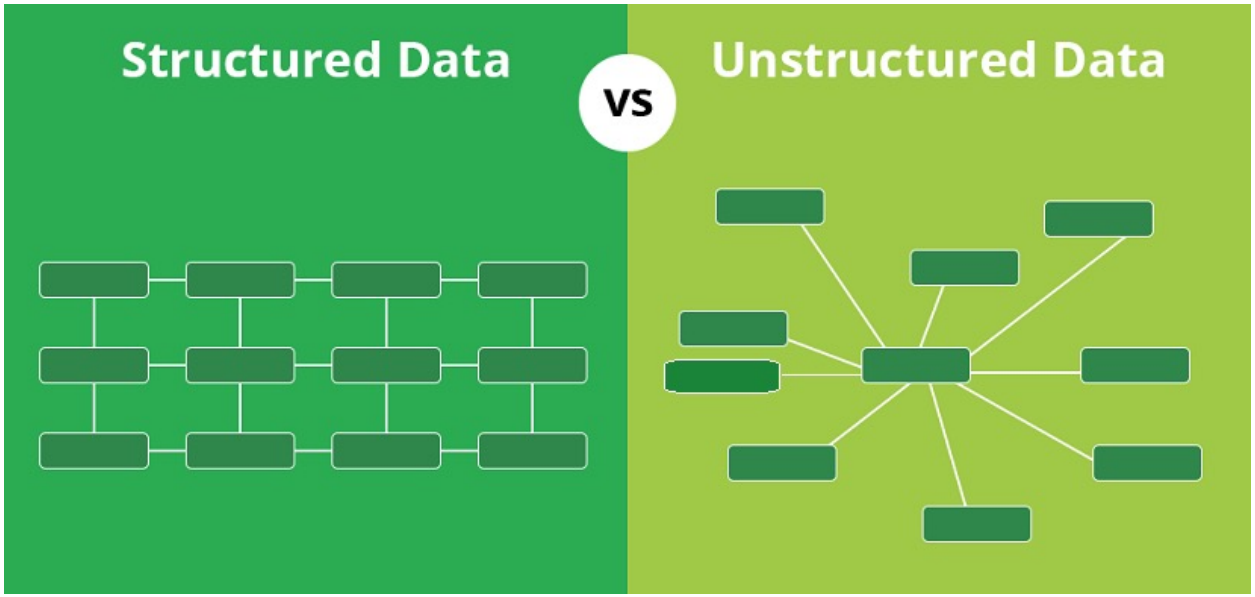
Data is growing by leaps and bounds every day. Some of it is **structured** (20%), but the large majority is **unstructured** (80-90%).



Structured Data and Unstructured Data

- **Data, the key resource in Data Science**

  Structured and unstructured data are **sourced**, **collected** and **scaled** in different ways, and each one resides in a different type of database.

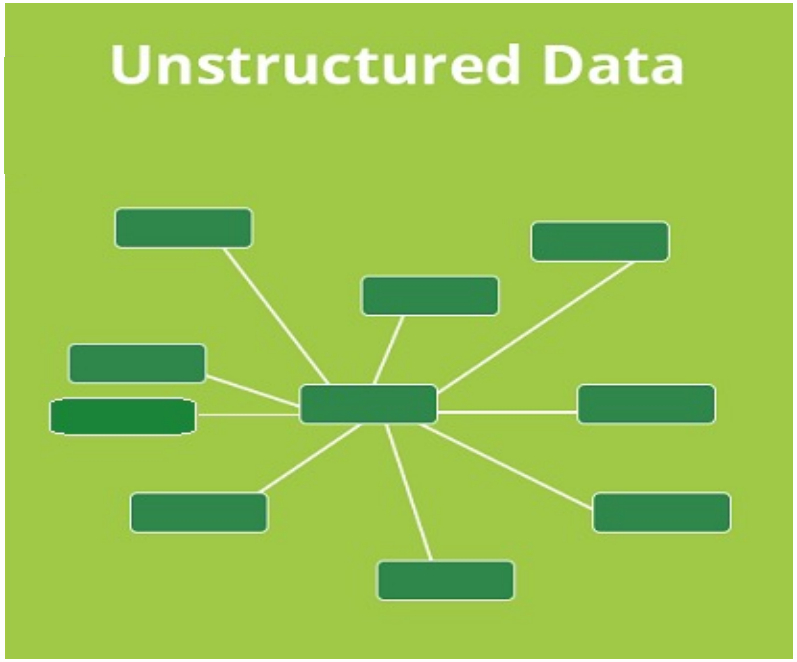


Unstructured Data

❑ Unstructured Data:

✓ It is data that isn't organized in a pre-defined fashion or lacks a specific data model.

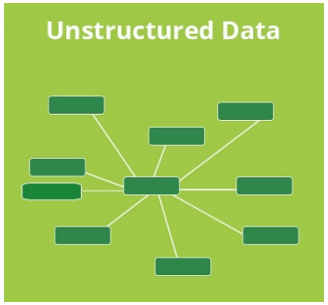
Unstructured data (e.g. text, mobile activity, social media posts, sensor data) can not be easily processed by a computer program.



Unstructured Data

❑ Unstructured Data Tools

- ✓ **MongoDB:** Uses flexible documents to process data for cross-platform applications and services.
- ✓ **DynamoDB**: Delivers single-digit millisecond performance at any scale via built-in security, in-memory caching and backup and restore.
- ✓ **Hadoop**: Provides distributed processing of large data sets using simple programming models and no formatting requirements.
- ✓ **Azure**: Enables agile cloud computing for creating and managing apps through Microsoft's data centers.



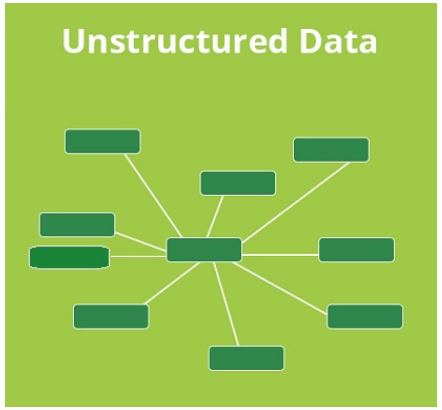
Unstructured Data

❑ Pros of Unstructured Data

- ✓ **Native format**: Stored in its native format, remains undefined until needed
- ✓ **Fast accumulation rates**: They can be collected quickly and easily
- ✓ **Data lake storage**: Allows for massive storage and pay-as-you-use pricing

❑Pros of Structure Data

- ✓ **Requires expertise**: Due to its non-formatted nature, DS expertise is required to prepare and analyze unstructured data
- ✓ **Limited storage options**: Specialized tools are required to manipulate.

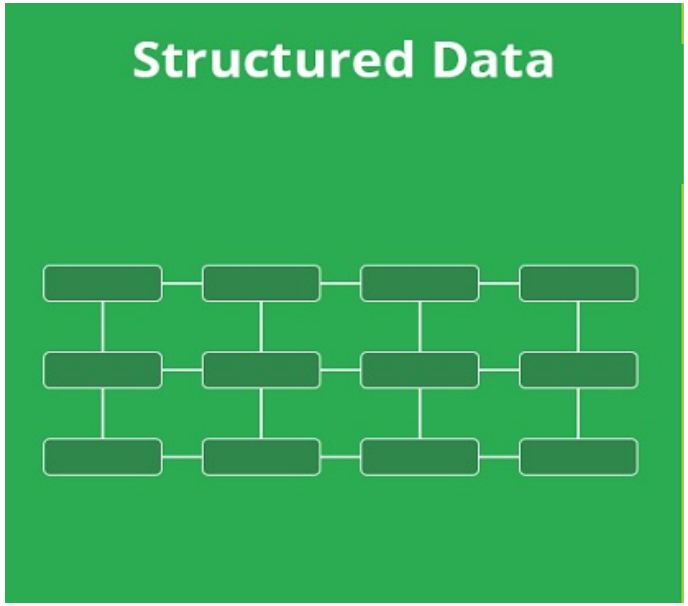


Structured Data

❑ Structure Data:

✓ It is data that has clear, definable relationships between the data points, with a pre-defined model containing it.

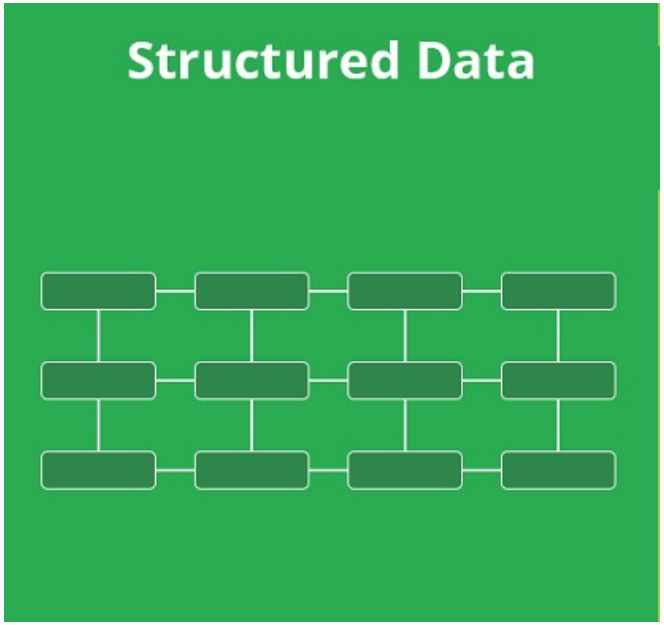
It is data which is in an organized form (e.g., in rows and columns) and can be easily used by a computer program.



Structured Data

❑ Pros of Structure Data

- ✓ Easily used by machine learning (ML) algorithms.
- ✓ Easily used by business users.
- ✓ Accessible by more tools



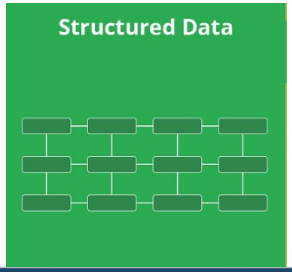
❑Pros of Structure Data

- ✓ Limited usage
- ✓ Limited storage options

Structured Data

❑ Structured Data Tools

- ✓ **OLAP**: Performs high-speed, multidimensional data analysis from unified, centralized data stores.
- ✓ **SQLite**: Implements a self-contained, serverless, zero-configuration, transactional relational database engine.
- ✓ **MySQL**: Embeds data into mass-deployed software, particularly mission-critical, heavy-load production system.
- ✓ **PostgreSQL**: Supports SQL and JSON querying as well as high-tier programming languages (C/C+, Java, Python, etc.).



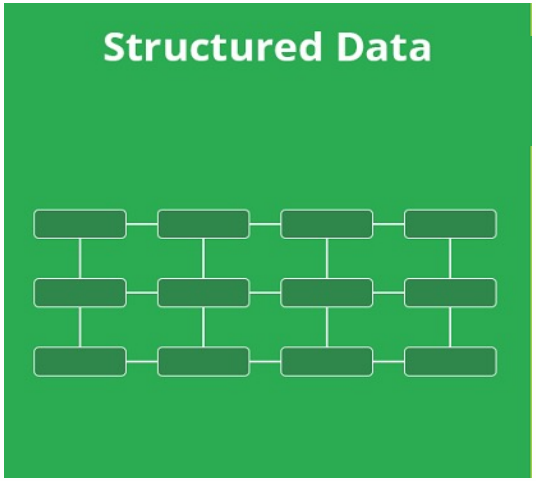
Structured Data

❑ Pros of Structure Data

- ✓ Easily used by machine learning (ML) algorithms.
- ✓ Easily used by business users.
- ✓ Accessible by more tools

❑Pros of Structure Data

- ✓ Limited usage
- ✓ Limited storage options



Structured Data Description

❑ Relational Database

✓ A relational database organizes data into **tables** which can be linked.

✓ The term "**Relation**" can be likened/interchanged with the term "**Tabular Data**".

✓ In a relational database, all data are **stored** and **accessed** via **relations**.

Structured Data Description

❑ Characteristics of a Relation

✓ **Rows** are called **tuples** (or **records**). A row represent a ***single instance*** of a relation, and must be **unique**.

ID	Last Name	First Name	Role
1	Kolter	Zico	Instructor
2	Manek	Gaurav	TA
3	Rice	Leslie	TA
4	Peres	Filipe	TA
5	Gates	William	Student
6	Musk	Elon	Student

Structured Data Description

❑ Characteristics of a Relation

✓ **Columns** are called ***attributes***, they specify some element contained by each of the ***tuples***.

ID	Last Name	First Name	Role
1	Kolter	Zico	Instructor
2	Manek	Gaurav	TA
3	Rice	Leslie	TA
4	Peres	Filipe	TA
5	Gates	William	Student
6	Musk	Elon	Student

Structured Data Description

❑ Multiple Tables/Relations

✓In practice, a database usually houses multiple tables/relations.

The tables have at least one column that could be used to establish a link between them.

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Role

ID	Name
1	Instructor
2	TA
3	Student

Structured Data Description

❑ Primary Keys

✓ **Primary Key**: A ***unique ID*** for every ***tuple*** in a relation (i.e. every row in the table), each relation must have exactly one primary key.

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Role

ID	Name
1	Instructor
2	TA
3	Student

Structured Data Description

❑ Foreign Keys

✓ A **Foreign Key** is an attribute that points to the primary key of another relation. If you delete a primary key, you need to delete all foreign keys pointing to it.

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Role

ID	Name
1	Instructor
2	TA
3	Student

Structured Data Description

❑ Indexes (not indices):

✓ **Indexes** are created as ways to "**quickly**" access elements of a **table**. For example, consider finding people with the ***last name*** "**Gates**": no option but just scan through the whole dataset: ***O (n)*** **operation**.

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Structured Data Description

❑ Indexes:

✓ Think of an index as a ***separate sorted table*** containing the ***indexed column*** and the ***tuple location***: searching for the value takes ***O(log n)*** time.

Person

Location	ID	Last Name	First Name	Role ID
0	1	Kolter	Zico	1
100	2	Manek	Gaurav	2
200	3	Rice	Leslie	2
300	4	Peres	Filipe	2
400	5	Gates	William	3
500	6	Musk	Elon	3

Last Name Index

Last Name	Location
Gates	400
Kolter	0
Manek	100
Musk	500
Peres	300
Rice	200

Structured Data Description

❑ Indexes:

✓ Without indexing, the search time is ***O (n).***

✓ *With indexing, the search time is* ***O(log n)****.*

For example, if *n = 100, then the search time without indexing would be = O(100)* → *100ms*

With indexing would be = O(log100) → *4.6052ms*

Structured Data Description

❑ Indexes:

- ✓ The **Primary Key** always has an index associated with it (so you can think of **Primary Keys** themselves as always being a fast way to access data).
- ✓ **Indexes** don't have to be on a single column, can have an index over multiple columns (with some ordering).

Structured Data Description

❑ Entity Relationships

Several types of inter-table relationships exist:

- ✓ One-to-one
- ✓ One-to-zero/one relationship
- ✓ One-to-many (and many-to-one)
- ✓ Many-to-many

Structured Data Description

❑ Entity Relationships

- ✓ These relate one (or more) rows in a table with one (or more) rows in another table, via a ***Foreign Key***.
- ✓ Note that these relationships are really between the "**entities**" that the tables represent, but we won't formalize this beyond the basic intuition.

Structured Data Description

❑ Entity Relationships

✓ One-to-many relationship: We have already seen a one-to-many relationship: **one role** can be shared by **many people**, denoted as follows.

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

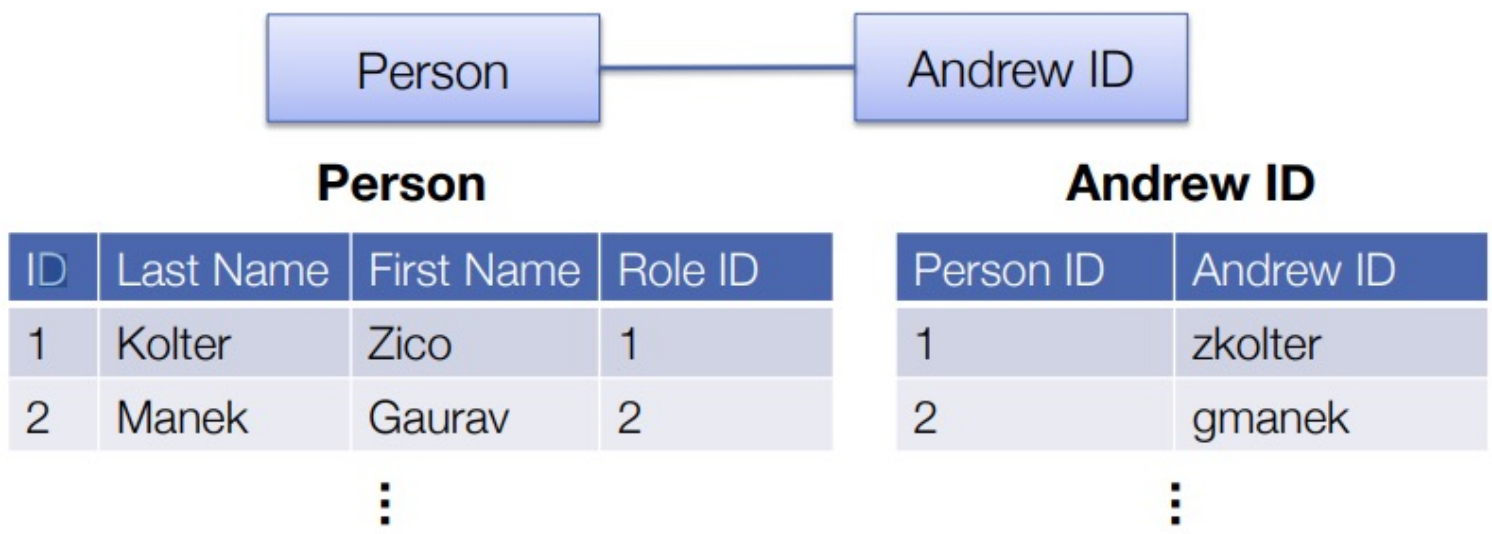
Role

ID	Name
1	Instructor
2	TA
3	Student

Structured Data Description

❑ Entity Relationships

✓ One-to-one relationship: In a true one-to-one relationship spanning multiple tables, **each row** in ***a table*** has **exactly** one row in ***another table***.



Structured Data Description

❑ Entity Relationships

- ✓ One-to-zero/one relationship: More common in databases is to find "one-to-zero/one" relationships broken across multiple tables.

- ✓ Consider adding a "**Grades**" table to our database: each person can have at most one tuple in the grades table.

Structured Data Description

❑ Entity Relationships

✓ One-to-zero/one relationship: Bars and circles denote "**mandatory**" versus "**option**" relationships (we won't worry about these, just know that there is notation for them).

Grades

Person ID	HW1 Grade	HW2 Grade
5	100	80
6	60	80

Structured Data Description

❑ Entity Relationships

✓ Many-to-many relationship: Creating a **Grades** table as done before is a bit cumbersome, because we need to keep adding columns to the table, null entries if someone doesn't do the homework.

Homework

ID	Name	388 Points	688 Points
1	HW 1	65	35
2	HW 2	75	25

Person Homework

Person ID	HW ID	Score
5	1	100
5	2	80
6	1	60
6	2	80

Structured Data Description

❑ Entity Relationships

✓ Many-to-many relationship:

Alternatively, consider adding two tables, a "homework" table that represents information about each homework, and an associative table that links homework to people

Homework

ID	Name	388 Points	688 Points
1	HW 1	65	35
2	HW 2	75	25

Person Homework

Person ID	HW ID	Score
5	1	100
5	2	80
6	1	60
6	2	80

Structured Data Description

❑ Associative Tables

✓ Associative tables: What is the primary key of this table? What are foreign keys?

✓ Which indexes would you want to create on this table?

Person Homework

Person ID	HW ID	Score
5	1	100
5	2	80
6	1	60
6	2	80

Structured Data Description

❑ Poll: Primary Key of Associative Table

✓ What should the primary key be for this table?

1. Person ID
2. HW ID
3. (Person ID, HW ID)
4. (Person ID, HW ID, Score)

Person Homework

Person ID	HW ID	Score
5	1	100
5	2	80
6	1	60
6	2	80

Structured Data Description

❑ Many-to-many Relationships

✓ Setups like this encode many-to-many relationships: each person can have multiple homeworks, and each homework can be done by multiple people.

Structured Data Description

❑ Many-to-many Relationships

✓ We could also write this in terms of relationships specified by the **associative table**, but this is ***not really correct***, as it is mixing up the underlying relationships with how they are stored in a database.

Data Manipulation

❑ Pandas

✓ Pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool.

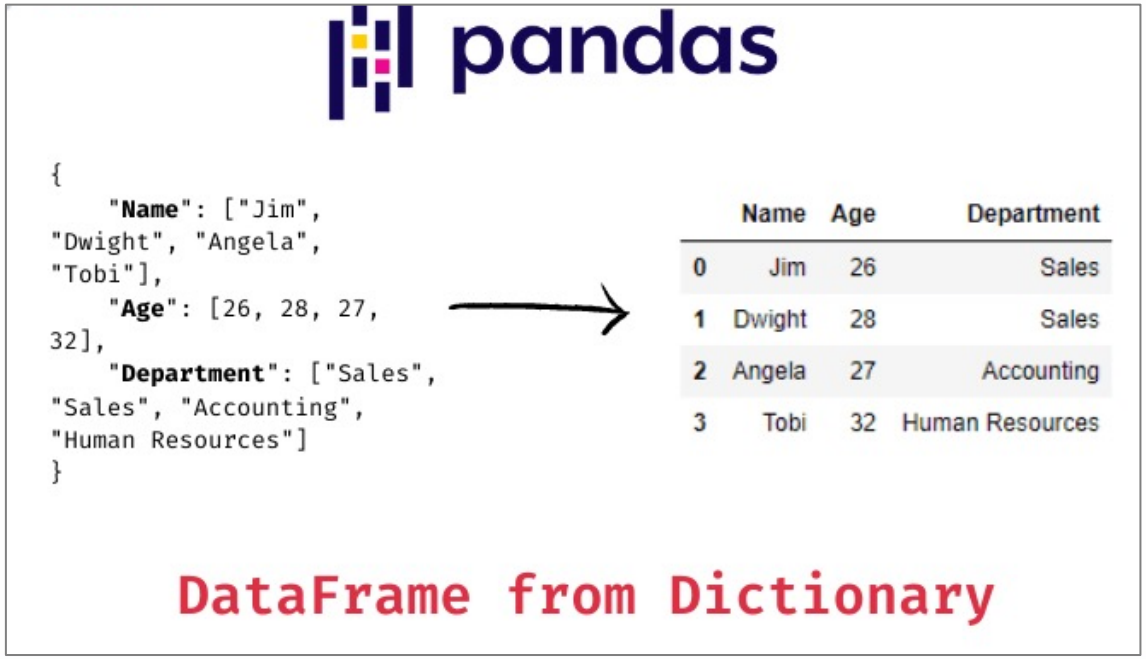
❑ SQLite

✓ SQLite is a RDBMS contained, not a client–server database engine though embedded into the end program

Data Manipulation with Pandas

❑ Pandas

✓ Pandas is a "Data Frame" library in Python, meant for manipulating in-memory data with **row** and **column labels** (as opposed to, e.g., matrices, that have no row or column labels).

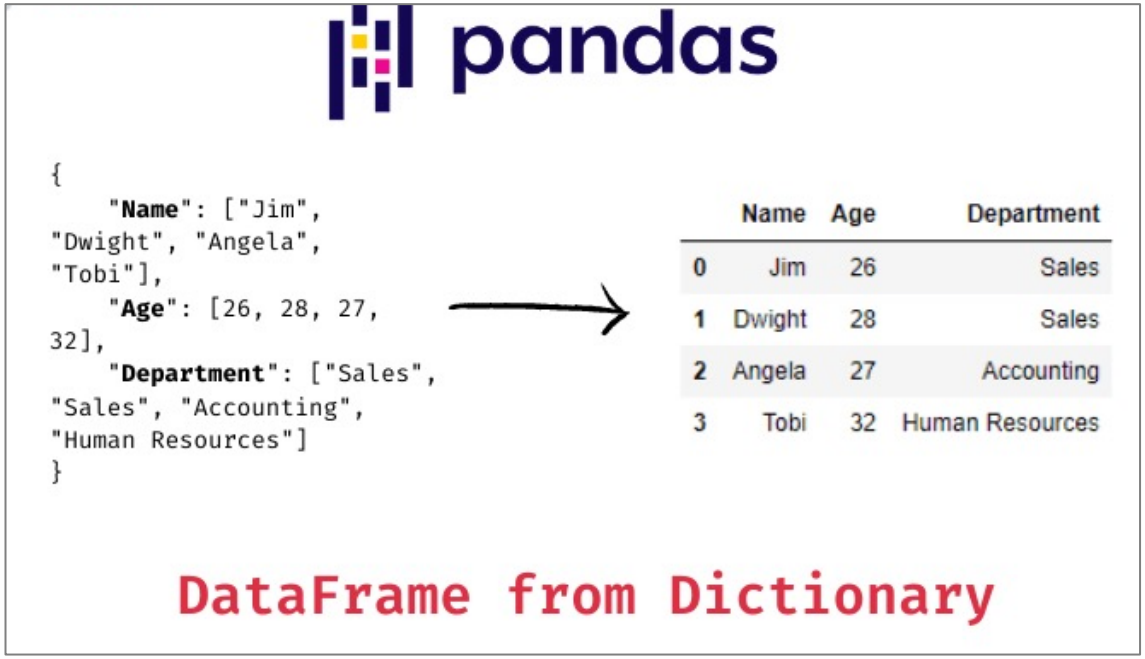


Data Manipulation with Pandas

❑ Pandas

✓ Pandas is not a relational database system, but it contains functions that mirror some functionality of relational databases.

We're going to cover Pandas in more detail in other portions of the class, but just discuss basic functionality for now.



Data Manipulation with Pandas

❑ Pandas examples

✓ Create a DataFrame with our **Person** example:

```
import pandas as pd

df = pd.DataFrame([(1, 'Kolter', 'Zico'),
                   (2, 'Manek', 'Gaurav'),
                   (3, 'Rice', 'Leslie'),
                   (4, 'Peres', 'Filipe'),
                   (5, 'Gates', 'Bill'),
                   (6, 'Musk', 'Elon')],
                  columns=["Person ID", "Last Name", "First Name"])
df.set_index("Person ID", inplace=True)
```

```
1 df
```

Person ID	Last Name	First Name
1	Kolter	Zico
2	Manek	Gaurav
3	Rice	Leslie
4	Peres	Filipe
5	Gates	Bill
6	Musk	Elon

Data Manipulation with Pandas

❑Important Notes:

- ✓ As mentioned, Pandas is not a **relational database system**, in particular it has no notion of **primary keys** (but it does have ***indexes***).
- ✓ Operations in Pandas are typically not in place (that is, they return a new modified DataFrame, rather than modifying an existing one).

  Use the "**inplace**" flag to make them done in place

Data Manipulation with Pandas

❑ Important Notes:

✓ If you select a single row or column in a Pandas DataFrame, this will return a "Series" object, which is like a one-dimensional DataFrame (it has only an index and corresponding values, not multiple columns)

Data Manipulation with Pandas

❑Common Pandas commands

```
# read CSV file into DataFrame
df = pd.read_csv(filename)

# get first five rows of DataFrame
df.head()

# index into a dataframe
# df.loc[rows, columns] and df.iloc[row numbers, column numbers]
df.loc[:, "Last Name"]            # Series of all last names
df.loc[:, ["Last Name"]]          # DataFrame with one column
df.loc[[1,2], :]                  # DataFrame with only ids 1,2
df.loc[1,"Last Name"] = "Kilter"  # Set an entry in a DataFrame
df.loc[7,:] = ("Moore", "Andrew") # Add a new entry with index=7
df.iloc[0,0]                      # Index rows and columns by zero-index
```

We’re going to cover more in next lecture in conjunction with data cleaning and so on.

Data Manipulation with SQLite

❑ SQLite

✓ An actual relational database management system (RDBMS).

✓ Unlike most systems, it is a serverless model, and applications directly connect to a file.

Allows for simultaneous connections from many applications to the same database file (but not quite as much concurrency as client-server systems).

Data Manipulation with SQLite

❑ SQLite

- ✓ All operations in SQLite will use SQL (Structured Query Language) command issued to the database object
- ✓ You can enforce foreign keys in SQLite, but we won't bother

Data Manipulation with SQLite

❑ Creating a Database and Table

✓ You can create a database and connect using this boilerplate code:

```
import sqlite3
conn = sqlite3.connect("people.db")
cursor = conn.cursor()

# do your stuff

conn.close()
```

✓ Create a new table:

```
cursor.execute("""
CREATE TABLE role (
    id INTEGER PRIMARY KEY,
    name TEXT
)""")
```

Data Manipulation with SQLite

❑ Creating a Database and Table

✓ Insert data into the table:

```
cursor.execute("INSERT INTO role VALUES (1, 'Instructor')")
cursor.execute("INSERT INTO role VALUES (2, 'TA')")
cursor.execute("INSERT INTO role VALUES (3, 'Student')")
conn.commit()
```

✓ Delete items from a table:

```
cursor.execute("DELETE FROM role WHERE id == 3")
conn.commit()
```

✓ Note: if you don't call commit, you can undo with conn.rollback()

Data Manipulation with SQLite

❑ Querying all Data from a Table

✓ Read all the rows from a table:

```
for row in cursor.execute('SELECT * FROM role'):
    print row
```

✓ Read table directly into a Pandas DataFrame:

```
pd.read_sql_query("SELECT * FROM role", conn, index_col="id")
```

id	name
1	Instructor
2	TA
3	Student

Manipulation with Pandas & SQLite

❑ Handling Joins

- ✓ Join operations merge multiple tables into a single relation (can be saved as a new table or just directly used)
- ✓ Four typical types of joins:

  1. **Inner** 2. **Left** 3. **Right** 4. **Outer**

- ✓ You join two tables on columns from each table, where these columns specify which rows are kept

Manipulation with Pandas & SQLite

❑ Example: Joining Person and Grades

✓ Consider joining two tables, Person and Grades, on ID / Person ID

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Manek	Gaurav	2
3	Rice	Leslie	2
4	Peres	Filipe	2
5	Gates	William	3
6	Musk	Elon	3

Grades

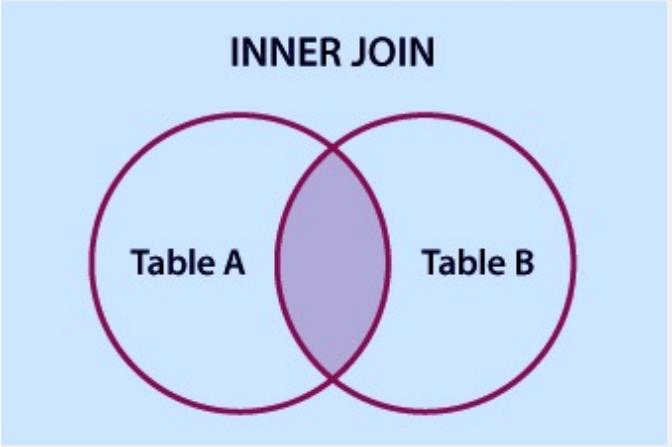
Person ID	HW1 Grade	HW2 Grade
5	100	80
6	60	80
100	100	100

Manipulation with Pandas & SQLite

❑ Inner Join (usually what you want)

✓ Join two tables where we only return the rows where the two joined columns contain the same value

Only these two rows have an entry in "Person" and an entry in "Grades"



ID	Last Name	First Name	Role ID	HW1 Grade	HW2 Grade
5	Gates	William	3	100	80
6	Musk	Elon	3	60	80

Manipulation with Pandas & SQLite

❑ Inner join in Pandas/SQLite

✓ In Pandas, you can also join on index using right_index/left_index parameters

```
# Pandas way
df_person = pd.read_sql_query("SELECT * FROM person", conn)
df_grades = pd.read_sql_query("SELECT * FROM grades", conn)
df_person.merge(df_grades, how="inner",
                left_on="id", right_on="person_id")

# SQLite way
cursor.execute("SELECT * FROM person, grades WHERE
                person.id == grades.person_id")
```

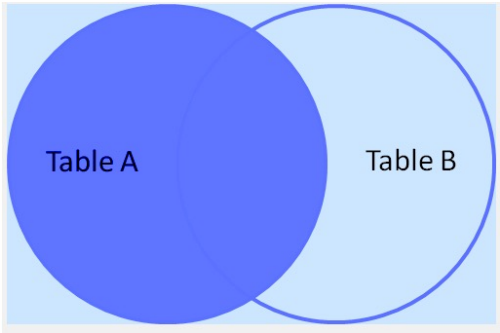
✓ There is also the join call in Pandas, which is a bit more limited (always assumes right is joined on index, left not on index)

Manipulation with Pandas & SQLite

❑ Left Joins

✓ Keep all rows of the left table, add entries from right table that match the corresponding columns.

Example: left join Person and Grades on ID, Person ID



ID	Last Name	First Name	Role ID	HW1 Grade	HW2 Grade
1	Kolter	Zico	1	NULL	NULL
2	Manek	Gaurav	2	NULL	NULL
3	Rice	Leslie	2	NULL	NULL
4	Peres	Filipe	2	NULL	NULL
5	Gates	William	3	100	80
6	Musk	Elon	3	60	80

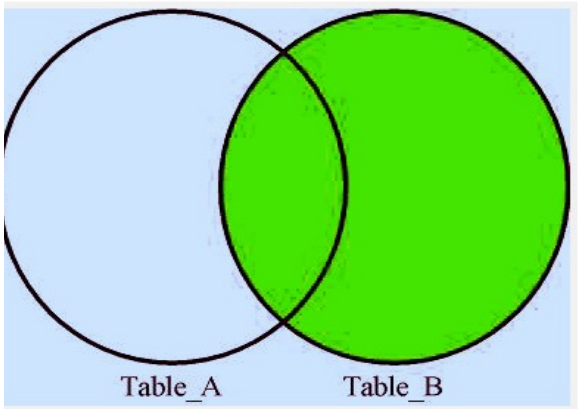
Manipulation with Pandas & SQLite

❑ Left Join in Pandas and SQLite

```
# Pandas way
df_person.merge(df_grades, how="left",
                left_on="id", right_on="person_id")

# SQLite way
cursor.execute("SELECT * FROM person LEFT JOIN grades ON
                person.id == grades.person_id")
```

	id	last_name	first_name	person_id	hw1_grade	hw2_grade
0	1	Kolter	Zico	NaN	NaN	NaN
1	2	Manek	Gaurav	NaN	NaN	NaN
2	3	Rice	Leslie	NaN	NaN	NaN
3	4	Peres	Filipe	NaN	NaN	NaN
4	5	Gates	Bill	5.0	85.0	95.0
5	6	Musk	Elon	6.0	80.0	60.0



Manipulation with Pandas & SQLite

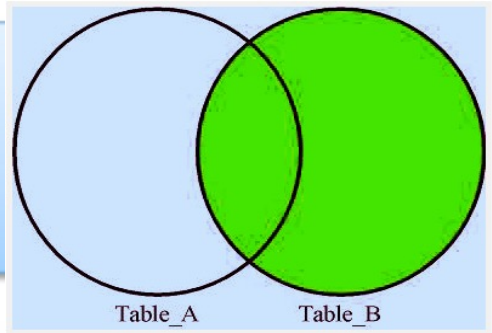
❑ Right Join

✓Like a left join but with the roles of the tables reversed

ID	Last Name	First Name	Role ID	HW1 Grade	HW2 Grade
5	Gates	William	3	100	80
6	Musk	Elon	3	60	80
100	NULL	NULL	NULL	100	100

```
# Pandas way
df_person.merge(df_grades, how="right",
                left_on="id", right_on="person_id")

# Not supported in SQLite
```



Manipulation with Pandas & SQLite

❑ Outer Join

✓Return all rows from both left and right join

ID	Last Name	First Name	Role ID	HW1 Grade	HW2 Grade
1	Kolter	Zico	1	NULL	NULL
2	Manek	Gaurav	2	NULL	NULL
3	Rice	Leslie	2	NULL	NULL
4	Peres	Filipe	2	NULL	NULL
5	Gates	William	3	100	80
6	Musk	Elon	3	60	80
100	NULL	NULL	NULL	100	100

```
# Pandas way
df_person.merge(df_grades, how="outer",
                left_on="id", right_on="person_id")
```

❑ Summary for today's lecture

We have learned about digital data description (structured and unstructured data) and fundamental manipulate approaches.

Questions and Comments!