DATA 200 Studies

By: Ryan Thompson, Liz Nguyen, Hannie Pham, Michelle Cao

Database Systems

Jupyter Notebook

GitHub

Database Systems: Organizes and structures data for retrieval and manipulation.

Data Providers & Clients:

Provider: Hosts and manages data (e.g., APIs, databases).

Client: Retrieves and processes data for analysis.

Jupyter Notebook

Web-based interactive environment for coding, documentation, and visualization.

Supports live code execution (Python, R, etc.), Markdown, and LaTeX for equations.

Key Features:

- Code cells and markdown support.
- Notebook server runs locally (`http://localhost:8888/tree`).
- Supports version control and sharing via GitHub.

GitHub

Version control system for tracking and collaborating on projects.

Key features:

- Repositories (repo) store code and files.
- Commit & Push: Save and upload changes.
- Pull Requests: Review and merge changes into main branches.

File System and Paths

File Systems: Organizes persistent data on storage devices.

Files contain:

- **Header**: Metadata (e.g., name, size).
- **Data**: Main content.
- **EOF (End-of-File)**: Marker for termination.

File Paths

Absolute Paths: Start from the root (e.g., `/home/user/docs`).

Relative Paths: Start from the current working directory (`./subfolder`).

- **Path Separators**:
  - Linux/Mac: /
  - Windows: \

Example - Python File Handling

`open("file.txt", "r")`: Open a file in read mode.

`os.path.join(dir, file)`: Construct file paths correctly.

Always close files after use: `file.close()`.

Introduction to Numpy

- NumPy is an acronym for “Numerical Python” and is the fundamental Python package for high performance computing and data analysis.
- At the core of the NumPy package is the ndarray object, which stands for “n-dimensional array.”
- NumPy arrays have a fixed size at creation, and are all required to be of the same kind of data.

Vectorization

Vectorization is the practice of replacing explicit loops with array expressions. In general, vectorized array operations will often be one to two (or more) orders of magnitude faster than their pure Python equivalents, with the biggest impact seen in any kind of numerical computations.

```
import numpy as np

# Create Python list of height (inch)
height = [72, 68, 69, 68, 64, 72, 72]

# Create Numpy version of height
np_height = np.array(height)

# Convert height to meters
np_height_m = np_height*0.0254

print(np_height_m)
```

```
[1.8288, 1.7271, 1.7526, 1.7271, 1.6256, 1.8288, 1.8288]
```

Question: What happens when you add NumPy Arrays?

They are Added Elementwise, while python lists are concatenated

```
python_list1 = [2,4,6]
python_list2 = [1,3,5]

numpy_list1 = np.array(python_list1)
numpy_list2 = np.array(python_list2)

print("Adding the Python lists gives " + str(python_list1 + python_list2))
print("Adding the Numpy lists gives " + str(numpy_list1 + numpy_list2))
```

```
Adding the Python lists gives [2, 4, 6, 1, 3, 5]
Adding the Numpy lists gives [ 3  7 11]
```

Indexing

Syntax	Result
`x[start:end]`	Elements in `x` indexed from `start` to `end` (but `end` is not included).
`x[start:end:step]`	Elements in `x` indexed from `start` to `end` using the step size `step`. Note that `end` is not included, and the default step value is 1.
`x[start:]`	Elements in x indexed from `start` through the end of the array.
`x[:end]`	Elements in x indexed from the beginning through `end` (but `end` is not included).

Let's Practice!

- age = np.array([18, 23, 89, 11, 35, 37, 26, 52, 76])
- What would these give us?
- What is the result of:
- age[2]
- age[2:5]
- age[4:]
- age[::-1]

NumPy. Arrange

Numpy.arrange(start, stop, step)

Default step is 1

```
print("The result of np.arange(1,8) is " + str(np.arange(1,8)))
print("The result of np.arange(2,10,2) is " + str(np.arange(2,10,2)))
print("The result of np.arange(2,10.1,2) is " + str(np.arange(2,10.1,2)))
print("The result of np.arange(5) is " + str(np.arange(5)))
print("The result of np.arange(-6,3) is " + str(np.arange(-6,3)))
print("The result of np.arange(5,1,-1) is " + str(np.arange(5,1,-1)))
```

```
The result of np.arange(1,8) is [1 2 3 4 5 6 7]
The result of np.arange(2,10,2) is [2 4 6 8]
The result of np.arange(2,10.1,2) is [  2.   4.   6.   8.  10.]
The result of np.arange(5) is [0 1 2 3 4]
The result of np.arange(-6,3) is [-6 -5 -4 -3 -2 -1  0  1  2]
The result of np.arange(5,1,-1) is [5 4 3 2]
```

Reshape vs Resize

- Reshape changes the NumPy array without changing the actual data
- Resize changes both the size and the shape of the data.

```
x = np.arange(1,10)
A = x.reshape(3,3)
print(x)
print(A)
print(x)
```

```
[1 2 3 4 5 6 7 8 9]

[[1 2 3]
 [4 5 6]
 [7 8 9]]

[1 2 3 4 5 6 7 8 9]
```

```
x = np.arange(1,10)
A = x.reshape(3,3)
print(x)
x.resize(3,3)
print(x)
```

```
[1 2 3 4 5 6 7 8 9]

[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

Function	Result
`numpy.amin(.)`	Determines the minimum value of the element along a specified axis.
`numpy.amax(.)`	Determines the maximum value of the element along a specified axis.
`numpy.mean(.)`	Determines the mean value
`numpy.median(.)`	Determines the median value
`numpy.std(.)`	Determines the standard deviation
`numpy.var(.)`	Determines the variance
`numpy.average(.)`	Determines a weighted average

Broadcasting

- Broadcasting allows you to perform arthmetic between ndarrays of different sizes
- Rules of Broadcasting from the Python Handbook:

Rules of Broadcasting:

Rule 1: If the two arrays differ in their number of dimensions, the shape of the one with fewer dimensions is padded with ones on its leading (left) side.

Rule 2: If the shape of the two arrays does not match in any dimension, the array with shape equal to 1 in that dimension is stretched to match the other shape.

Rule 3: If in any dimension the sizes disagree and neither is equal to 1, an error is raised.

```
matrix = np.arange(12).reshape(3,4)
vector = np.array([5,6,7]).reshape(3,1)


print(matrix)
print(vector)

matrix += vector

print()
print('The updated matrix is')
print(matrix)
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]]

[[5]
 [6]
 [7]]

The updated matrix is
[[ 5  6  7  8]
 [10 11 12 13]
 [15 16 17 18]]
```

Introduction to Pandas

- What is Pandas?
  - Open-source data manipulation and analysis library in Python
  - Built of Numpy, optimized for fast, vectorized operations
  - Handles structured data similar to SQL tables or spreadsheets

- Why Pandas?
  1. Fast and flexible for tabular data
  2. Handles missing data efficiently
  3. Seamless integration with SQL, big data, and cloud storage
  4. Built-in functions for aggregation and transformation

Basic Pandas Data Structures

- Series (1D Data)

```python
import pandas as pd
s = pd.Series([10, 20, 30],
              index=['A', 'B', 'C'])
print(s)
```

```
A    10
B    20
C    30
dtype: int64
```

DataFrame (2D Data)

```python
df = pd.DataFrame({'Name': ['Alice', 'Bob'],
                   'Age': [25, 30]})
print(df)
```

```
    Name  Age
0  Alice   25
1    Bob   30
```

Data Access & Manipulation

- Accessing Data Efficiently
  - Column selection

```
df['Age']  # Single column
df[['Name', 'Age']]  # Multiple columns
```

  - Row selection – Label-based (.loc[]) vs. Position-based (.iloc[])
  - Filtering rows

```
df[df['Age'] > 25]  # Condition-based filtering
df.query("Age > 25")  # Alternative (faster for large data)
```

- Optimizing Large DataFrames
  - Categorical data types

```
df['Category'] = df['Category'].astype('category')
```

  - Chunked data loading

```
for chunk in pd.read_csv('bigfile.csv', chunksize=50000):
    process(chunk)
```

- Indexing for Faster Lookups

```
df.set_index('Name', inplace=True)
df.loc['Alice']  # Fast lookup
```

Merging & Combining DataFrames

- Merging allows combining datasets based on key columns

Merge Type	Description	Example Use Case
Inner Join	Returns **only matching rows** from both DataFrames	`df1.merge(df2, on='ID', how='inner')`
Left Join	Returns **all rows from left**, and matching rows from right	`df1.merge(df2, on='ID', how='left')`
Right Join	Returns **all rows from right**, and matching rows from left	`df1.merge(df2, on='ID', how='right')`
Outer Join	Returns **all rows from both**, filling missing values	`df1.merge(df2, on='ID', how='outer')`

- concat() Concatenation - stacks DataFrames **vertically or horizontally**.

Imagine

- First day of work as a Data Analyst at ShopEase, Monday morning, coffee in hand, boss walks in
- “I need to know which of our registered customers have made purchases. Can you get me a report ASAP?”
    a. Inner join
    b. Left join
    c. Right join
    d. Outer join

Customers DataFrame:

	Customer_ID	Name
0	101	Alice
1	102	Bob
2	103	Charlie
3	104	David

Sales DataFrame:

	Customer_ID	Purchase
0	101	Laptop
1	103	Phone
2	105	Tablet

Imagine

- But just as you're about to take a sip of your coffee, an email pops up
- “Actually, I also need to see ALL registered customers, even if they haven’t made a purchase yet. Can you add that?”
    a. Inner join
    b. Left join
    c. Right join
    d. Outer join

Customers DataFrame:

	Customer_ID	Name
0	101	Alice
1	102	Bob
2	103	Charlie
3	104	David

Sales DataFrame:

	Customer_ID	Purchase
0	101	Laptop
1	103	Phone
2	105	Tablet

Imagine

- About to step away for lunch, your Teams notification pings
- “You know what, I need to see all transactions, even from guest checkouts, whether they’re registered or not?”
    a. Inner join
    b. Left join
    c. Right join
    d. Outer join

Customers DataFrame:

	Customer_ID	Name
0	101	Alice
1	102	Bob
2	103	Charlie
3	104	David

Sales DataFrame:

	Customer_ID	Purchase
0	101	Laptop
1	103	Phone
2	105	Tablet

Imagine

- Boom, you're feeling like a data magician. One final request tho…
- "Just give me EVERYTHING. All registered customers, all purchases, whether they match or not. I want the full picture."
    a. Inner join
    b. Left join
    c. Right join
    d. Outer join

Customers DataFrame:

	Customer_ID	Name
0	101	Alice
1	102	Bob
2	103	Charlie
3	104	David

Sales DataFrame:

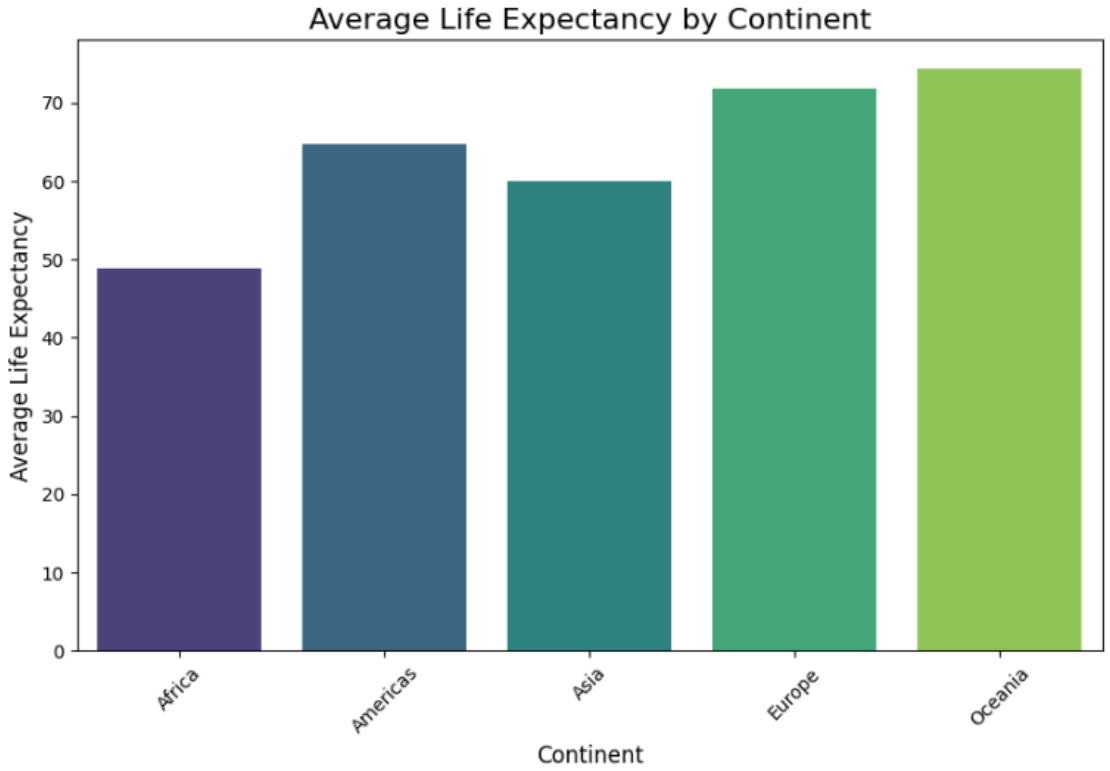
	Customer_ID	Purchase
0	101	Laptop
1	103	Phone
2	105	Tablet

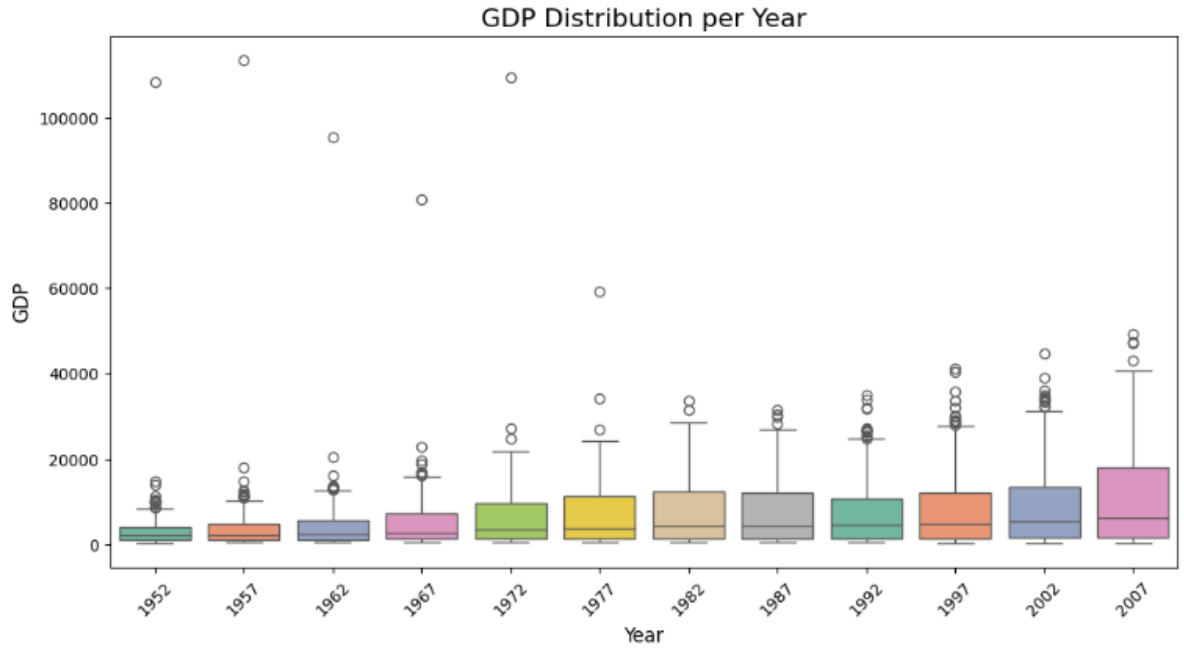
Visualization with Matplotlib

- Matplotlib is a powerful Python library for data visualization.
- Used for creating static, animated, and interactive visualizations.
- Works well with NumPy, Pandas, and other data analysis libraries.
- Key module: `matplotlib.pyplot`
- `Use %matplotlib notebook` for interactive plots
- Use plt.savefig() to save plots
  - Parameters to specify resolution, size, type of file, etc.

Types of Plots

- Line Plot (`plt.plot()`) – Used for trends over time
- Bar Chart (`plt.bar()`) – Used for categorical comparisons
- Histogram (`plt.hist()`) – Used for showing distributions
- Scatter Plot (`plt.scatter()`) – Used for relationships between variables
- Box Plot (`plt.boxplot()`) – Used for statistical summaries

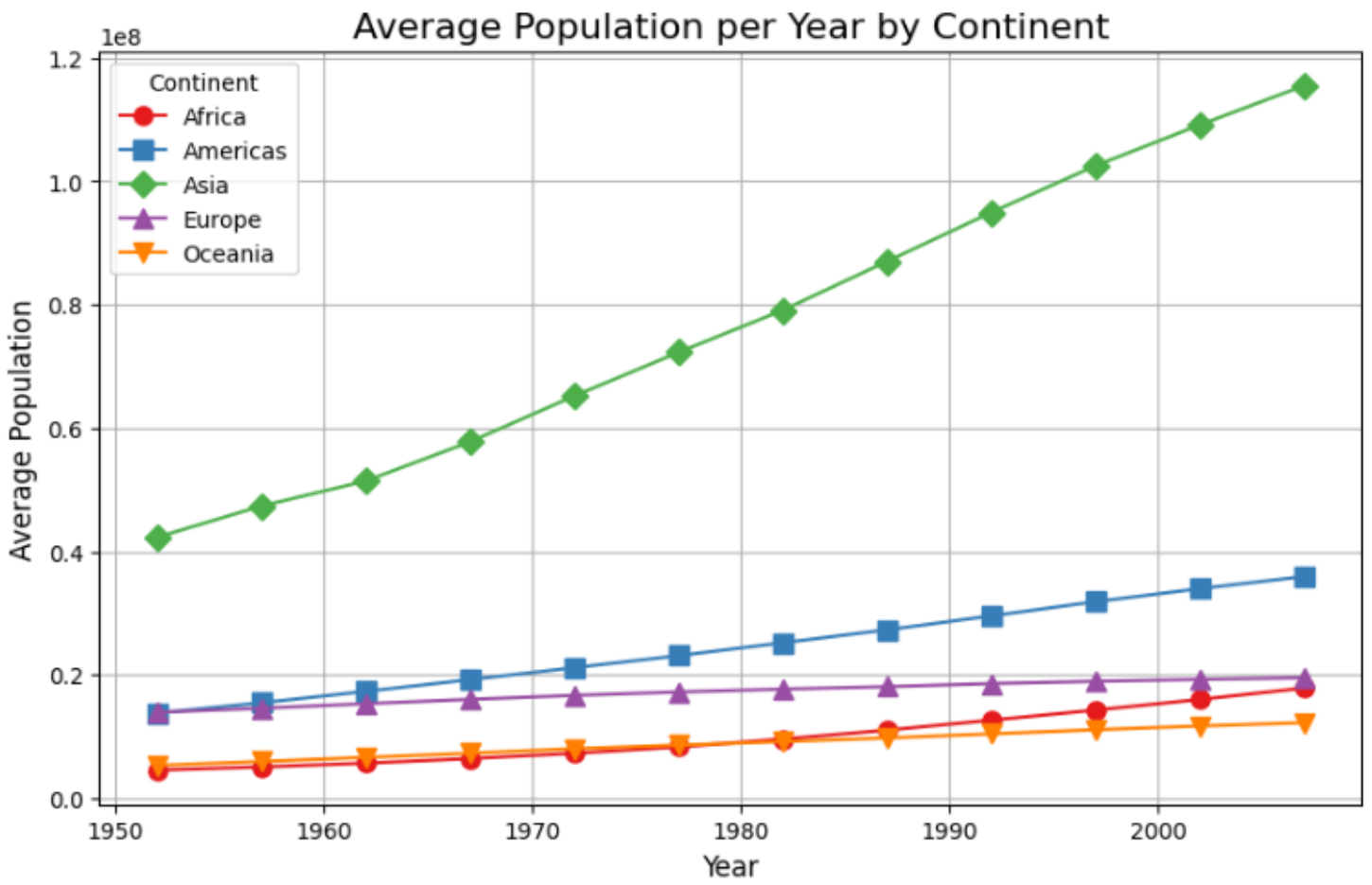




Derived from countries.csv

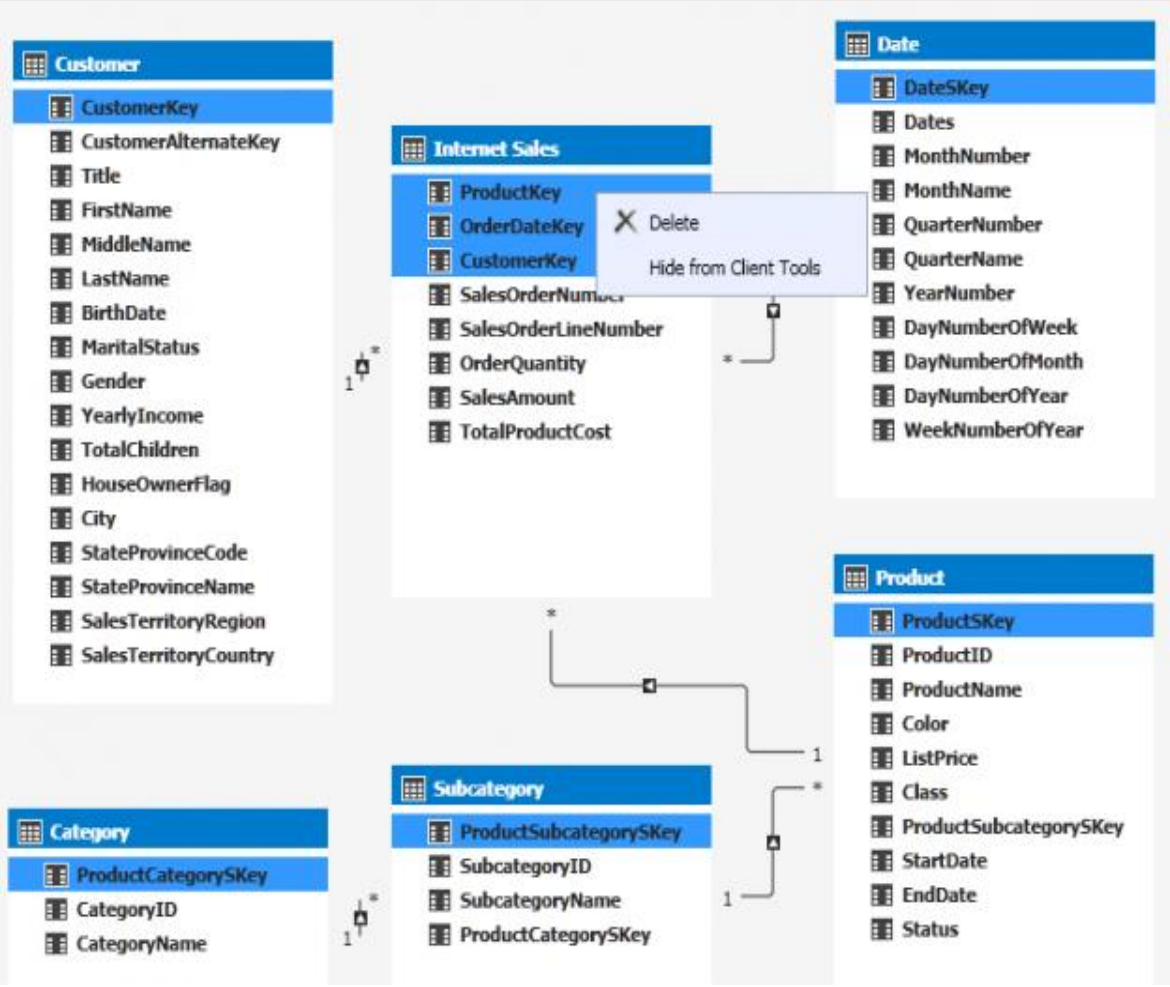
Customize Plots

- **Color & Line Style** – `color='red', linestyle='--'`
- **Markers** – `marker='o', markersize=8`
- **Figure Size** – `plt.figure(figsize=(6,4))`
- **Title & Labels** – `plt.title(), plt.xlabel(), plt.ylabel()`
- **Grid & Legend** – `plt.grid(), plt.legend()`



Derived from countries.csv

Tabular Models



🌟 Definition

A structured data model that organizes information into rows and columns (like a spreadsheet).

Each row represents a record (data entry).

Each column represents an attribute (variable).

🌟 Key Characteristics

Fixed schema: Each column has a defined data type (e.g., integer, string, date).

Constraints: No duplicate rows, consistent column data types.

Operations: Filtering, sorting, merging, and modifying tables.

🌟 Tabular Model in Data Analysis

Efficient for structured data storage (e.g., CSV, Excel, SQL databases).

Common tools:

- Pandas (Python): `df = pd.read_csv("data.csv")`
- SQL Queries: `SELECT * FROM table WHERE condition`

Relational Database/Models

- A relational Model is a way to store data into a relation or a table.
- Relational databases are based on the relational model, which is a way of representing data in tables, as mentioned above.
- In the relational database model, the structure consists of multiple, inter-dependent tables
  - each table is comprised of rows and columns, just like in tabular model.

Keys

- A **key** is one or more fields (columns) that can be used to uniquely identify an individual record (row).
- Primary Key: Uniquely identifies each row within a table.
  - Ex: employee_id in Employees table
- Foreign Key: a column (or group of columns) in a relational database that provide a link between data in two tables.
  - Team_id in a players table, which also exists in Teams table

Relationships in the Relational Model

- **One-to-One (1:1)**
  - Each row in Table A relates to exactly one row in Table B.
  - **Example**: A person and their passport.
- **One-to-Many (1:N)**
  - One row in Table A relates to multiple rows in Table B.
  - **Example**: A manager can oversee multiple employees.
- **Many-to-Many (M:N)**
  - Rows in Table A relate to multiple rows in Table B, and vice versa. This is often handled with a **junction table**.
  - **Example**: Players and teams—each player can play for multiple teams in different seasons, and each team has multiple players.

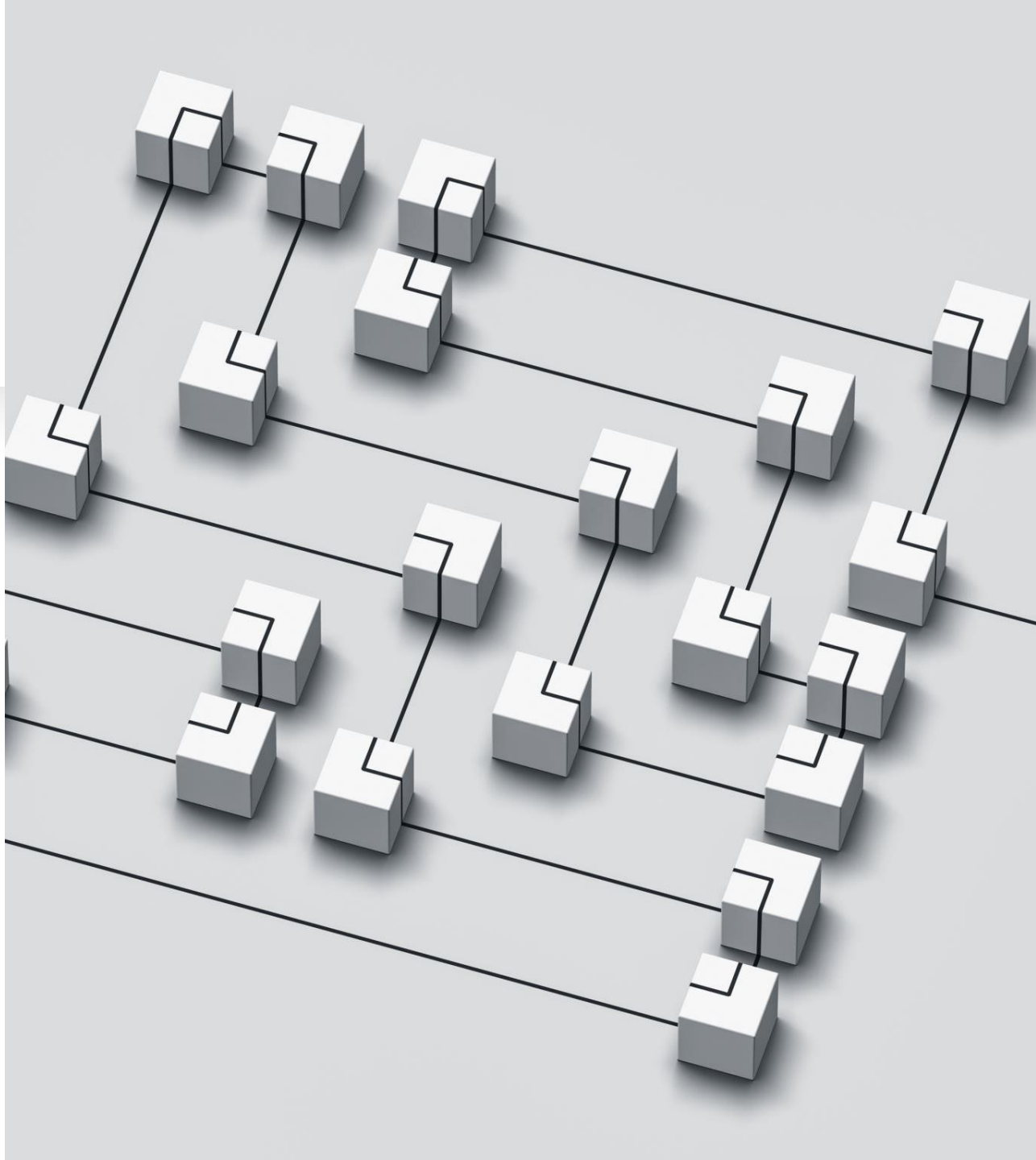
Coding with Relational Models (SQL)

- Relational Models are usually used using Structured Query Language (SQL)
- It is used to create, read, update, and delete data.

Command	Description	Example
`SELECT`	Retrieve data from a table	`SELECT * FROM employees;`
`WHERE`	Filter records based on conditions	`SELECT * FROM employees WHERE salary > 50000;`
`ORDER BY`	Sort result set in ascending or descending order	`SELECT first_name FROM employees ORDER BY salary DESC;`
`INSERT`	Insert new records into a table	`INSERT INTO employees (first_name, salary) VALUES ('John', 60000);`
`UPDATE`	Modify existing records in a table	`UPDATE employees SET salary = 65000 WHERE last_name = 'Doe';`
`GROUP BY`	Group rows that have the same values in specified columns	`SELECT department, AVG(salary) FROM employees GROUP BY department;`
`HAVING`	Filter groups of rows after grouping (used with `GROUP BY`)	`SELECT department, AVG(salary) FROM employees GROUP BY department HAVING AVG(salary) > 60000;`
`WHERE`	Specify a condition to filter individual rows	`SELECT * FROM employees WHERE department = 'Sales';`
`DELETE`	Remove records from a table	`DELETE FROM employees WHERE last_name = 'Doe';`

Joins

- A **JOIN** in SQL is used to combine rows from two or more tables based on a related column.
- There are 3 main types of Joins, Inner, Left, and Right.



Inner Join

- Returns only the rows where there is a match in both tables.
- Example:

Example:

sql Copy Edit

```sql
SELECT Users.username, Orders.product
FROM Users
INNER JOIN Orders ON Users.user_id = Orders.user_id;
```

Result:

username	product
John	Laptop
Alice	Phone

Left Join/Right Join

- A left join would return all rows from the left table, and the matching rows from the right table.
- A right join would return all rows from the right table, and the matching rows from the right table.

SQL Examples:

How would I select all columns from a table called Houses?

Answer

- Select * From Houses
- If you wanted to select certain columns and not all, it would be:
- Select column1, column2, From Houses
- Limit 5 at the end to limit to just 5 rows.

Return rooms column ordered by lightbulbs

Answer

- Select Rooms FROM Houses; Order By num_light_bulbs DESC
- (Or ASC) if you wanted in Ascending order
- Additional Info:
- Could also do:
- Select Rooms FROM Houses WHERE num_light_bulbs >, <, <=, >=, = any number.

Select Houses in New York with 3 Rooms

Answer

- Select * FROM Houses WHERE city = 'New York' AND num_rooms = 3.
- Can use this type of filter clause to search for anything within the House table.

Hierarchical Models

- What are Hierarchical Models?
  - A tree-like structure where one parent node can have multiple child nodes
  - Why important? Used in databases, APIs, web scraping, and data analysis
- Key characteristics
  - One-to-many relationships (parent-child structure)
  - Efficient querying for nested data
  - Common in JSON, XML, HTML, and directory structures
- Examples of Hierarchical Data:
  - Website structures
  - API responses (JSON/XML)
  - File systems
  - Financial transactions
  - Scientific measurements

JSON & XML in Hierarchical Models

- JSON
  - Lightweight, widely used for APIs and web data
  - Pros – readable, integrates with JavaScript
  - Cons – No built-in schema validation

```json
{
  "company": {
    "name": "TechCorp",
    "employees": [
      {"name": "Alice", "role": "Engineer"},
      {"name": "Bob", "role": "Manager"}
    ]
  }
}
```

- XML
  - Older format, but still used in data exchange
  - Pros – schema validation, widely supported
  - Cons – More verbose than JSON

```xml
<company>
  <name>TechCorp</name>
  <employees>
    <employee><name>Alice</name><role>Engineer</role></employee>
    <employee><name>Bob</name><role>Manager</role></employee>
  </employees>
</company>
```

Querying Hierarchical Data (XPath & Python)

- How to extract hierarchical data?
  - XPath: a query language for selecting nodes in an XML document
  - Python libraries: json, xml.etree.ElementTree, lxml
- XPath Queries for XML Data
  - Find GDP of France in 2017
  - Find all pop under France

```
/ind2/FRA/y2017/gdp
```

```
/ind2/FRA/*/pop
```

- Parsing JSON in Python

```python
import json
data = '{"company": {"name": "TechCorp", "employees": [{"name": "Alice"}, {"name": "Bob"}]}}'
parsed = json.loads(data)
print(parsed["company"]["employees"][0]["name"])  # Output: Alice
```

- Parsing XML in Python

```python
import xml.etree.ElementTree as ET
tree = ET.parse("data.xml")
root = tree.getroot()
for employee in root.findall(".//employee"):
    print(employee.find("name").text)
```

APIs

- API = application program interface
- Structured way of retrieving data
  - **Request:** `GET /weather?city=NewYork`
  - **Response:** `{ "temperature": 25, "humidity": 60, "description": "Clear sky" }`
- Most "popular" websites have APIs (eg: Google Maps, Twitter, Facebook)
- Does not violate legal/ethical code

Web Scraping

- Extract data from websites by HTML content
- Access publicly available data at any site (though bots may give you trouble)
- Data quality may not be as good
- Ethical/legal issue depending on use

Web Scraping cont'd

- `Selenium` package acts as a 'webdriver' or automates web interactions

```
[ ]  # scraping a single page
     comments=[]
     for x in comment_ids:
         userid_element = driver.find_element('xpath','//*[@id="' + x + '"]/div/div[2]/div[1]/span[1]/a[2]')
         userid = userid_element.text

         user_date = driver.find_element('xpath','//*[@id="' + x + '"]/div/div[2]/div[2]/span/a/time')
         date = user_date.get_attribute('title')

         user_comment = driver.find_element('xpath','//*[@id="' + x + '"]/div/div[3]/div/div[1]')
         comment = user_comment.text

         comments.append([userid,date,comment])
```

```
df_single = pd.DataFrame(comments,columns=['userid','date','comment'])
df_single
```

	userid	date	comment
0	merc1	March 24, 2002 9:54PM	I personally think that with a few tweaks the ...
1	fredvh	March 24, 2002 11:06PM	I am debating a new purchase and these two are...
2	blueguydotcom	March 25, 2002 9:02AM	Great handling, RWD, excellent engine and the ...
3	hungrywhale	March 25, 2002 3:04PM	And no manual tranny. That may not matter to y...
4	riez	March 25, 2002 4:44PM	One beauty of BMW 3 Series is that there are s...
5	blueguydotcom	March 26, 2002 12:20PM	good grief, so you wait 9 months for the manua...
6	hungrywhale	March 26, 2002 1:02PM	I'll give it a fair shot when the manual comes...
7	blueguydotcom	March 26, 2002 1:45PM	I understand it's not about the speed. I reall...
8	hungrywhale	March 26, 2002 4:00PM	It sounds like you know why I won't even put i...
9	riez	March 26, 2002 6:00PM	Is there such a thing as an automatic-only tru...
10	wishnhigh1	March 26, 2002 6:55PM	If I had to rank the entry lux performance sed...
11	shipo	March 26, 2002 9:06PM	Hey my friend, dig this, in Australia, the ONL...
12	mbarto	March 27, 2002 6:20AM	>"Is there such a thing as an automatic-only t...
13	hungrywhale	March 27, 2002 9:00AM	I think what riez and I are trying to say is n...
14	blueguydotcom	March 27, 2002 10:10AM	amen
15	dave330i	March 27, 2002 10:23AM	In this class, the best bang for the buck has ...
16	blueguydotcom	March 27, 2002 11:06AM	Even at 29k for a 35k msrp IS300, I passed on ...