Unit 2

Data Sources and APIs

Contents

Contents

Structured Data

- Structured data has a clear format, usually tabular.
- Examples: CSV, Excel, SQL tables.

Unstructured Data

- Unstructured data has no predefined format.
- Examples: text files, logs.

Reading a Text File

- Reading a text file in Python is simple.
- Using open() and read()
- Using open() and readlines()

Reading a Text File – Using open() and read()

```
f = open("example.txt", "r")
content = f.read()
print(content)
f.close()
```

✓ 0.0s

```
Hello
World
Bye
```

Reading a Text File – Using open() and read()

```
with open("example.txt", "r") as f:
    content = f.read()
    print(content)
```

✓ 0.0s

```
Hello
World
Bye
```

Reading a Text File – Using open() and readlines()

```
with open("example.txt", "r") as f:
    lines = f.readlines()
    print(lines)
```

✓ 0.0s

```
['Hello\n', 'World\n', 'Bye']
```

Writing to a Text File

- We can create a new file or overwrite/append an existing one.

```
f = open("example.txt", "w")
f.write("Hello World Bye")
f.close()
```

```
with open("example.txt", "w") as f:
    f.write("First line\n")
    f.write("Second line\n")
```

Reading/Writing from/to a csv file

- Using the csv module
- Using pandas

Using the csv module

```
import csv

with open("data.csv", "r") as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

✓ 0.0s

```
['Name', 'Address']
['Harry Potter', ' Hogwarts']
['Gandalf', ' Middle Earth']
```

Using the csv module

```
with open("data.csv", "r") as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(row["Name"], row["Address"])
```

✓ 0.0s

```
Harry Potter  Hogwarts
Gandalf  Middle Earth
```

csv.DictReader lets you read rows as dictionaries

Using the csv module

```python
import csv

data = [
    ["Name", "Age", "City"],
    ["Alice", 25, "NY"],
    ["Bob", 30, "LA"]
]

with open("output.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerows(data)
```

Using the csv module

```
import csv

data = [
    {"Name": "Alice", "Age": 25},
    {"Name": "Bob", "Age": 30}
]

with open("output.csv", "w", newline="") as f:
    writer = csv.DictWriter(f, fieldnames=["Name", "Age"])
    writer.writeheader()
    writer.writerows(data)
```

csv.DictWriter writes dictionaries

Using the pandas module

```
import pandas as pd

df = pd.read_csv("data.csv")
print(df)
```

✓ 2.7s

```
          Name        Address
0  Harry Potter       Hogwarts
1       Gandalf   Middle Earth
```

Using the pandas module

```
import pandas as pd

dataframe = pd.DataFrame([
    {"Name": "Alice", "Age": 25},
    {"Name": "Bobb", "Age": 30}
])

dataframe.to_csv("output.csv", index=False)
```

Reading a JSON File

A JSON file is a file that stores data in JSON format (JavaScript Object Notation) which is text-based, human-readable, and widely used for data exchange.

```
import json

with open("data.json", "r") as f:
    data = json.load(f)

print(data)
```

✓ 0.0s

```
{'name': 'Bob', 'age': 30, 'skills': ['Python', 'ML']}
```

Writing to a JSON File

```python
import json

data = {
    "name": "Bob",
    "age": 30,
    "skills": ["Python", "ML"]
}

with open("data.json", "w") as f:
    json.dump(data, f, indent=4)
```

Task: Excel File Manipulation

Objective: Read an Excel file, process the data, and write the results to a new Excel file.

Instructions:

You are provided with an Excel file named students.xlsx containing the following columns:

Name

Math

Science

English

Write a Python program that:

a. Reads the Excel file into a DataFrame (or using openpyxl if preferred).

b. Calculates the Total and Average marks for each student.

c. Adds these as new columns in the DataFrame or Excel sheet.

d. Identifies students who scored average >= 60 as "Passed" and < 60 as "Failed". Add a column Result for this.

e. Writes the updated data to a new Excel file students_result.xlsx.

Contents

Database Access with RDBMS

- **Relational Databases (RDBMS) store data in tables with rows and columns.**
- Data is structured and uses SQL (Structured Query Language).
- Relationships between tables are defined with keys (primary key, foreign key).
- Examples: MySQL, PostgreSQL, SQLite, Oracle, SQL Server.

Accessing RDBMS in Python – Using sqlite3

```
import sqlite3

# Step I - Connect to a database (or create it)
conn = sqlite3.connect("example.db")
cursor = conn.cursor()
```

Accessing RDBMS in Python – Using sqlite3

```
# Step II - Create table
cursor.execute("""
CREATE TABLE IF NOT EXISTS students(
    id INTEGER PRIMARY KEY,
    name TEXT,
    age INTEGER
)
""")
```

```
# Step III - Insert data
cursor.execute("INSERT INTO students (name, age) VALUES (?, ?)", ("Alice", 25))
conn.commit()
```

Accessing RDBMS in Python – Using sqlite3

```
# Step IV - Query data
cursor.execute("SELECT * FROM students")
rows = cursor.fetchall()
for row in rows:
    print(row)

conn.close()
```

✓ 0.0s

```
(1, 'Alice', 25)
```

Accessing RDBMS in Python

- We can also use other RDBMS (like MySQL or PostgreSQL)
- Use libraries like mysql-connector-python, psycopg2, or SQLAlchemy.
- SQL syntax is similar but the connection strings differ.

Using SQLAlchemy

- SQLAlchemy is a Python library that helps interact with databases using two approaches:
  - Core (SQL expression language): Writing SQL-like queries in Python.
  - ORM (Object-Relational Mapping): Working with Python classes instead of tables directly.

```
pukarkarki@macbookpro ~ % pip3 install sqlalchemy
```

Using SQLAlchemy: SQLite Example

```
from sqlalchemy import create_engine

engine = create_engine('sqlite:///mydb.db', echo=True)
```

✓ 3.2s

Using SQLAlchemy: SQLite Example

```
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy import Column, Integer, String

Base = declarative_base()

class Student(Base):
    __tablename__ = 'students'
    id = Column(Integer, primary_key=True)
    name = Column(String)
    age = Column(Integer)

# Create the table
Base.metadata.create_all(engine)
```

Defining a Table (ORM)

Using SQLAlchemy: SQLite Example

- **Creating a Session**

```
from sqlalchemy.orm import sessionmaker

Session = sessionmaker(bind=engine)
session = Session()


✓  0.0s
```

Session handles all the interactions with the database.

Using SQLAlchemy: SQLite Example

- **Adding Data**

```
student1 = Student(name="Alice", age=20)
student2 = Student(name="Bob", age=22)

session.add(student1)
session.add(student2)
session.commit()
```

Using SQLAlchemy: SQLite Example

- **Querying Data**

```
# Get all students
students = session.query(Student).all()
for s in students:
    print(s.name, s.age)

# Filter
alice = session.query(Student).filter_by(name="Alice").first()
print(alice.name, alice.age)
```

✓ 0.0s

Task

- Create a table for Books with fields: title, author, year.
- Insert 3 books and query by author.
- Update a book’s year.
- Delete a book.

Database Access with Non-Relational Databases

- Non-Relational Databases (NoSQL) store data in flexible formats.
- Useful for unstructured or semi-structured data.
- Can be
  - Document-based: JSON-like documents (MongoDB, CouchDB)
  - Key-Value: Simple key-value pairs (Redis, DynamoDB)
  - Column-family: Wide-column stores (Cassandra, HBase)
  - Graph: Nodes and edges (Neo4j)

Accessing NoSQL in Python

```
# Connecting to MongoDB
from pymongo import MongoClient

client = MongoClient("mongodb://localhost:27017/")
db = client["school"]          # Database
collection = db["students"]  # Collection (like a table)
```

Accessing NoSQL in Python

```python
# Inserting single document
student = {"name": "Alice", "age": 25, "subjects": ["Math", "Science"]}
collection.insert_one(student)
```

✓ 0.1s

```
InsertOneResult(ObjectId('6935a5a6845b57c7a1ff1b09'), acknowledged=True)
```

```python
# Inserting multiple document
students = [
    {"name": "Bob", "age": 30, "subjects": ["English", "History"]},
    {"name": "Carol", "age": 22, "subjects": ["Math", "Art"]}
]
collection.insert_many(students)
```

✓ 0.0s

Accessing NoSQL in Python

```
# Query - Find all students
for data in collection.find():
    print(data)
```

```
# Query - Find by Condition
for student in collection.find({"age": {"$gt": 24}}):  # age > 24
    print(student)
```

Accessing NoSQL in Python

```python
from pymongo import MongoClient

# Connect to MongoDB server
client = MongoClient("mongodb://localhost:27017/")

# List all databases
databases = client.list_database_names()
print(databases)
```

✓ 0.0s

```
['admin', 'confession', 'config', 'local', 'school']
```

```python
from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")
db = client["school"]              # select database
collection = db["students"]        # select collection

# Delete the collection
collection.drop()

print(db.list_collection_names())  # confirm it's deleted
```

✓ 0.0s

```
[]
```

```python
from pymongo import MongoClient

# Connect to MongoDB
client = MongoClient("mongodb://localhost:27017/")

# Delete a database named "school"
client.drop_database("school")

# Verify deletion
print(client.list_database_names())
```

✓ 0.0s

```
['admin', 'confession', 'config', 'local']
```

Contents

API

- API (Application Programming Interface) allows programs to communicate with other software.
- Common types: REST and SOAP.

REST APIs

- REST = Representational State Transfer.
- Uses HTTP methods: GET, POST, PUT, DELETE.
- Data is usually in JSON format.

```python
import requests
from pprint import pprint

# Example API endpoint
url = "https://jsonplaceholder.typicode.com/posts/1"

response = requests.get(url)  # GET request
if response.status_code == 200:
    data = response.json()  # convert JSON response to Python dict
    pprint(data)
else:
    print("Error:", response.status_code)
```

✓ 0.0s

```
{'body': 'quia et suscipit\n'
         'suscipit recusandae consequuntur expedita et cum\n'
         'reprehenderit molestiae ut ut quas totam\n'
         'nostrum rerum est autem sunt rem eveniet architecto',
 'id': 1,
 'title': 'sunt aut facere repellat provident occaecati excepturi optio '
          'reprehenderit',
 'userId': 1}
```

REST APIs

```python
import requests
url = "https://jsonplaceholder.typicode.com/posts"
payload = {"title": "Test", "body": "Hello API", "userId": 1}
response = requests.post(url, json=payload)
print(response.json())
```

✓ 0.8s

```
{'title': 'Test', 'body': 'Hello API', 'userId': 1, 'id': 101}
```

SOAP APIs

- SOAP = Simple Object Access Protocol.
- Uses XML for requests and responses.
- Requires a WSDL (Web Services Description Language) URL.

Accessing SOAP API using Python

```python
from zeep import Client

# WSDL URL
wsdl = "http://www.dneonline.com/calculator.asmx?WSDL"
client = Client(wsdl=wsdl)

# Call a SOAP method
result = client.service.Add(10, 20)
print(result)  # 30
```

✓ 3.2s

```
30
```

Accessing SOAP API using Python

```python
from zeep import Client

# WSDL URL
wsdl = "http://www.dneonline.com/calculator.asmx?WSDL"
client = Client(wsdl=wsdl)

# Call a SOAP method
result = client.service.Add(10, 20)
print(result)  # 30
```

✓ 3.2s

```
30
```

zeep library parses XML and allows calling methods like normal Python functions.

Which is Right for You?

Difference	SOAP	REST
Style	Protocol	Architectural style
Function	Function-driven: transfer structured information	Data-driven: access a resoruce for data
Data format	Only uses XML	Permits many data formats, including plain text, HTML, XML, and JSON
Security	Supports WS-Security and SSL	Supports SSL and HTTPS
Bandwidth	Requires more resources and bandwidth	Requires fewer resources and is lightweight
Data cache	Can not be cached	Can be cached
Payload handling	Has a strict communication contract and needs knowledge of everything before any interaction	Needs no knowledge of the API
ACID compliance	Has built-in ACID compliance to reduce anomalies	Lacks ACID compliance

Processing API Data

- Once you get the data:
- JSON : convert to Python dict/list, process with loops, filters, or pandas.
- XML : convert to dict using xmltodict or process with ElementTree.

Processing API Data using Pandas

```python
import requests

api_key = "YOUR_API_KEY"
city = "Kathmandu"
url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api_key}&units=metric"

response = requests.get(url)
data = response.json()
print(data)  # see the JSON structure
```

Processing API Data using Pandas

```
import pandas as pd
from pandas import json_normalize

# Flatten nested JSON
df = json_normalize(data)
print(df.head())
```

Processing API Data using Pandas

```
df_weather = df[[
    'name',              # city name
    'main.temp',         # temperature
    'main.humidity',     # humidity
    'wind.speed',        # wind speed
    'weather'            # weather description (list)
]]

# Extract weather description from the list
df_weather['weather_desc'] = df_weather['weather'].apply(lambda x: x[0]['description'])
df_weather.drop(columns='weather', inplace=True)

print(df_weather)
```

Contents

What is web scraping?

- Web scraping is the process of collecting data from websites.
- A program visits a webpage, reads the HTML, and extracts the information you want.
- This helps when data is not available through an API.

Why do we use web scraping?

- To collect data for analysis
- To track prices, weather, news, research data
- To gather information from websites that update often
- To automate data collection instead of doing it manually

How websites work?

- A webpage is usually written in HTML.
- It has tags like , <p>, <a>, and so on.
- When you open a page, your browser downloads this HTML.
- In scraping, you do the same thing, but with Python.

Main Steps in Scraping

- Two main steps

1) Download the webpage
   - Use the requests library. It sends an HTTP request to the website and receives HTML as text.

2) Parse the HTML
   - Use BeautifulSoup to read the HTML and extract the exact parts you want. You can search for tags, classes, ids, tables, links, etc.

HTTP Basics

- Web scraping uses HTTP.
- Common status codes:
- 200 → success
- 404 → page not found
- 403 → access denied
- 500 → server error
- If the code is not 200, scraping usually fails.

Limitations and Ethics

- Some websites block scraping.
- Always check the robots.txt file and the site’s policies.
- Don’t overload the server with too many requests.
- Use scraping only for legal and fair purposes.

Web Scraping Using requests and BeautifulSoup

- Install the libraries

pip install requests beautifulsoup4

```python
import requests

url = "https://www.scrapethissite.com/pages/simple/"
response = requests.get(url)

print(response.status_code)   # 200 means ok
print(response.text)          # raw HTML
```

✓ 1.5s

```
200
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Countries of the World: A Simple Example | Scrape This Site | A public sandb
    <link rel="icon" type="image/png" href="/static/images/scraper-icon.png" />
```

Parse HTML with BeautifulSoup

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(response.text, "html.parser")
```

Find elements

- You can search by tag name.

```
titles = soup.find_all("h1")
for t in titles:
    print(t.text)
```

✓ 0.0s

```
Countries of the World: A Simple Example
250 items
```

Find elements

- Search by class:

```
items = soup.find_all("p", class_="content")
for item in items:
    print(item.text)
```

Extract links

```
links = soup.find_all("a")
for link in links:
    print(link.get("href"))
```

✓ 0.0s

```
/
/pages/
/lessons/
/faq/
/login/
/lessons/
http://peric.github.io/GetCountries/
```

Save Scraped Data

```python
import csv

with open("links.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["Title", "Link"])
    for link in links:
        writer.writerow([link.text.strip(), link.get("href")])
```

```
Title,Link
Scrape This Site,/
Sandbox,/pages/
Lessons,/lessons/
FAQ,/faq/
Login,/login/
4 video lessons,/lessons/
http://peric.github.io/GetCountries/,http://peric.github.io/GetCountries/
```

Contents

Handling Large Datasets

- Large datasets often do not fit into memory.
- If you try to load a huge CSV or process a massive file at once, your program may slow down or crash.
- To avoid this, we use **chunking** and **lazy evaluation.**

Handling Large Datasets - Chunking

- Chunking means loading a large dataset in small parts instead of loading it all at once.
- Example idea:

  - Instead of reading a 5 GB CSV file at once, you read it in parts of 10,000 rows each.

Process one chunk → free memory → load next chunk.

Handling Large Datasets - Chunking

- **Why ?**
- Saves memory
- Works even on slow or low-RAM systems
- Helps when you need to process data in a loop
- Makes data pipelines more stable

Handling Large Datasets - Chunking

```python
import pandas as pd

for chunk in pd.read_csv("big_file.csv", chunksize=10000):
    print(chunk.head())
```

```python
import pandas as pd
import os

input_file = "big_file.csv"
output_file = "processed.csv"

first_chunk = True

for chunk in pd.read_csv(input_file, chunksize=10000):
    # process the chunk
    # example: keep rows where value > 0
    # chunk = chunk[chunk["value"] > 0]

    chunk.to_csv(
        output_file,
        mode="w" if first_chunk else "a",
        header=first_chunk,
        index=False
    )

    first_chunk = False
```

Handling Large Datasets - Lazy Evaluation

- Lazy evaluation means delay actual computation until the moment it is needed.
- You don’t load everything up front.
- You set up the work, but Python only performs it when you ask for a result.

Handling Large Datasets - Lazy Evaluation

- Where?
- Python generators
- Iterators
- Libraries like dask
- Reading files line-by-line
- Streaming data

Handling Large Datasets - Lazy Evaluation

```python
def read_lines(filename):
    with open(filename) as f:
        for line in f:
            yield line  # created only when needed

for line in read_lines("example.txt"):
    print(line)
```

✓ 0.0s

```
First line

Second line
```


Basic example with a generator

Review Questions

- What is structured data? Give two examples.
- What is unstructured data? Give two examples.
- How do you read a CSV file in Python using pandas?
- How do you write a DataFrame to a JSON file in Python?
- What is the difference between reading a text file with open() and reading a CSV with pandas?
- What problems may occur when reading messy CSV files?
- Explain how encoding (like UTF-8) affects reading text files.

Review Questions

- What is the main difference between a relational database and a non-relational database?
- What is SQL, and why is it important?
- Why do some applications prefer NoSQL databases?
- What Python libraries can you use to connect to a SQL database?
- What is a collection in MongoDB?

Review Questions

- What is an API?
- What is the difference between REST and SOAP APIs?
- What does “HTTP GET request” mean?
- How do you read JSON data returned by a REST API in Python?
- What is an API key and why is it used?
- What is the typical format of a REST API response?
- Why is error handling important when calling APIs?

Review Questions

- What is web scraping?
- Why should you check a website’s robots.txt file before scraping?
- What does the requests library do in web scraping?
- What is BeautifulSoup used for?
- What does soup.find() return?
- What are common problems you may face while scraping?
- Why do some websites block scraping?
- What are ethical concerns related to web scraping?

Review Questions

- Why can’t large datasets be loaded into memory at once?
- What is chunking?
- What is lazy evaluation?
- How do generators help with memory efficiency?
- Give one example where chunking is useful.
- Explain the difference between chunking and lazy evaluation.