


MODULE / WEEK 04

DATA EXTRACTION PROCESS


QMB4400
DATA ANALYSIS AND OPTIMIZATION



1

LIVE CLASSROOM


- Lecture – Data Extraction
 - PYTHON
 - JSON
 - SQLite



2

HISTORY OF JSON

- JSON – JavaScript Object Notation
- Quickly become the de factor standard for information exchange
- Transport some data from here to there
- Use JSON to store and exchange data?



3

JSON

```
{
  "firstName": "Jane",
  "lastName": "Doe",
  "hobbies": ["running", "sky diving", "singing"],
  "age": 35,
  "children": [
    {
      "firstName": "Alice",
      "age": 6
    },
    {
      "firstName": "Bob",
      "age": 8
    }
  ]
}
```



4

JSON

- Java support JSON natively
- JSON is a built-in package in Python
 - For encoding and decoding JSON data
 - Import json



5

SERIALIZING JSON

Simple Python objects are translated to JSON according to a fairly intuitive conversion

Python	JSON
dict	Object
list, tuple	Array
str	String
Int, long, float	number
True	true
False	false
None	null



6

SIMPLE SERIALIZATION EXAMPLE

```
data = {
    "president": {
        "name": "Zaphod Beeblebrox",
        "species": "Betelgeusian"
    }
}

with open("data_file.json", "w") as write_file:
    json.dump(data, write_file)

json_string = json.dumps(data)
```



7

SOME USEFUL KEYWORD ARGUMENTS

Both the `dump()` and `dumps()` methods use the same keyword arguments.

```
>>> json.dumps(data)
>>> json.dumps(data, indent=4)
```



8

DESERIALIZING JSON

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
Null	None



9

EXAMPLE

- The simplest example would be encoding a tuple and getting back a list after decoding, like so:

```
>>> blackjack_hand = (8, "Q")
>>> encoded_hand = json.dumps(blackjack_hand)
>>> decoded_hand = json.loads(encoded_hand)

>>> blackjack_hand == decoded_hand
False
>>> type(blackjack_hand)
<class 'tuple'>
>>> type(decoded_hand)
<class 'list'>
>>> blackjack_hand == (decoded_hand)
True
```



10

SIMPLE DESERIALIZATION EXAMPLE

```
with open("data_file.json", "r") as read_file:
    data = json.load(read_file)

    json_string = """
    {
      "researcher": {
        "name": "Ford Prefect",
        "species": "Betelgeusian",
        "relatives": [
          {
            "name": "Zaphod Beeblebrox",
            "species": "Betelgeusian"
          }
        ]
      }
    }
    """
    data = json.loads(json_string)
```



11

REAL WORLD EXAMPLE (sort of)

This example will use JSONPlaceholder, a great source of fake JSON data. We will be working on a list of TODOs.

- Create a script file called scratch.py.
- Add these two (2) imports at the top of your file:


```
import json
import requests
```
- An API request to the JSONPlaceholder service is needed, or just use the requests package to do the heavy lifting. The request is for the /todos endpoint.


```
response = requests.get("https://jsonplaceholder.typicode.com/todos")
todos = json.loads(response.text)
```



12

REAL WORLD EXAMPLE (sort of)

Run the file in interactive mode and test it for yourself. Check the type of todos and look at the first 10.

```
>>> todos == response.json()
True
>>> type(todos)
<class 'list'>
>>> todos[:10]
...
```



13

REAL WORLD EXAMPLE (sort of)

What is the interactive mode?

- Jump back and forth between your editor and the terminal
 - Use the `-i` interactive flag when you run the script
 - Great trick for testing code because it runs the script and then opens up an interactive command prompt with access to all the data from the script

You can see the structure of the data by visiting the endpoint in a browser:

JSON

```
{
  "userId": 1,
  "id": 1,
  "title": "delectus aut autem",
  "completed": false
}
```



14

REAL WORLD EXAMPLE (sort of)

```
# Map of userId to number of complete TODOs for that user
todos_by_user = {}
```

```
# Increment complete TODOs count for each user.
```

```
for todo in todos:
    if todo["completed"]:
        try:
```

```
            # Increment the existing user's count.
```

```
            todos_by_user[todo["userId"]] += 1
```

```
except KeyError:
```

```
    # This user has not been seen. Set their count to 1.
```

```
    todos_by_user[todo["userId"]] = 1
```

```
• # Create a sorted list of (userId, num_complete) pairs.
```



15

REAL WORLD EXAMPLE (sort of)

```
# Create a sorted list of (userId, num_complete) pairs.
top_users = sorted(todos_by_user.items(),
                    key=lambda x: x[1], reverse=True)
# Get the maximum number of complete TODOs.
max_complete = top_users[0][1]

# Create a list of all users who have completed
# the maximum number of TODOs.
users = []
for user, num_complete in top_users:
    if num_complete < max_complete:
        break
    users.append(str(user))

max_users = " and ".join(users)
```



16

REAL WORLD EXAMPLE (sort of)

Python:

```
>>> s = "s" if len(users) > 1 else ""
>>> print(f"user{s} {max_users} completed {max_complete}
TODOs")
users 5 and 10 completed 12 TODOs
```



17

REAL WORLD EXAMPLE (sort of)

Python:

```
# Define a function to filter out completed TODOs
# of users with max completed TODOs.
def keep(todo):
    is_complete = todo["completed"]
    has_max_count = str(todo["userId"]) in users
    return is_complete and has_max_count

# Write filtered TODOs to file.
with open("filtered_data_file.json", "w") as data_file:
    filtered_todos = list(filter(keep, todos))
    json.dump(filtered_todos, data_file, indent=2)
```



18

ENCODING & DECODING CUSTOM PYTHON OBJECTS

Python:

```
class Elf:
    def __init__(self, level, ability_scores=None):
        self.level = level
        self.ability_scores = {
            "str": 11, "dex": 12, "con": 10,
            "int": 16, "wis": 14, "cha": 13
        } if ability_scores is None else ability_scores
        self.hp = 10 + self.ability_scores["con"]
```



19

SIMPLIFYING DATA STRUCTURES

Python:

```
>>> elf = Elf(level=4)
>>> json.dumps(elf)
TypeError: Object of type 'Elf' is not JSON serializable
```



20

SIMPLIFYING DATA STRUCTURES

Python:

```
>>> z = 3 + 8j
>>> type(z)
<class 'complex'>
>>> json.dumps(z)
TypeError: Object of type 'complex' is not JSON serializable
```

When a real number and an imaginary number love each other very much, they add together to produce a number which is (justifiably) called complex.



21

SIMPLIFYING DATA STRUCTURES

Python:

```
>>> z.real
```

```
3.0
```

```
>>> z.imag
```

```
8.0
```

Passing the same numbers into a complex constructor is enough to satisfy the `__eq__` comparison operator:

Python:

```
>>> complex(3, 8) == z
```

```
True
```



22

ENCODING CUSTOM TYPES

```
json_str = json.dumps(4+6j, default=complex_encoder)
```

Need to define `complex_encoder()`, which will convert our complex object into a tuple (which is serializable)

```
def complex_encoder(z):
```

```
    if isinstance(z, complex):
```

```
        return (z.real, z.imag)
```

```
    else:
```

```
        type_name = z.__class__.__name__
```

```
        raise TypeError(f"Object of type {type_name} is not serializable")
```



23

ENCODING CUSTOM TYPES

We could also achieve this by subclassing `json.JSONEncoder`

```
class ComplexEncoder(json.JSONEncoder):
```

```
    def default(self, z):
```

```
        if isinstance(z, complex):
```

```
            return (z.real, z.imag)
```

```
        else:
```

```
            return super().default(z)
```

If we choose this method instead, default wont work. We need to use `cls` instead.

```
json_str = json.dumps(4+6j, cls=ComplexEncoder)
```



24

DECODING CUSTOM TYPES FROM JSON

We can represent a complex object in JSON like this

PYTHON:

```
{
    "__complex__": true,
    "real": 42,
    "imaginary": 36
}
```

If we let the `load()` method deserialize this, we'll get a Python dict instead of our desired complex object. That's because JSON objects deserialize to Python dict. We can write a custom decoder function that will read this dictionary and return our desired complex object.



25

DECODING CUSTOM TYPES FROM JSON

PYTHON:

```
def decode_complex(dct):
    if "__complex__" in dct:
        return complex(dct["real"], dct["imaginary"])
    else:
        return dct
```



26

DECODING CUSTOM TYPES FROM JSON

Now, we need to read our JSON file and deserialize it. We can use the optional `object_hook` argument to specify our decoding function.

PYTHON:

```
with open("complex_data.json") as complex_data:
    z = json.load(complex_data, object_hook=decode_complex)
```

Now, if we print the type of `z`, we'll see

PYTHON:

```
<class 'complex'>
```



27

SQLite

SQLite is included with every python install so it is as simple as
import sqlite3.

Next we connect to an sqlite database –
db_filename = '/tmp/example.db'
connection = sqlite3.connect(db_filename)

Create a simple table:
connection.execute("""
CREATE TABLE IF NOT EXISTS person (
first_name TEXT,
last_name TEXT,
age FLOAT
)
""")



28

SQLite

Insert myself the author into the database:
connection.execute("""
INSERT INTO person (first_name, last_name, age)
VALUES (?, ?, ?)
""", ('tina', 'harclerode', 18))
connection.commit()

We can query using a simple select statement. But notice that it only
returns tuples. This is not very efficient as we would like to get the
column names as well.

```
for row in connection.execute('SELECT * FROM person'):
    print(row)
```



29

SQLite

```
('tina', 'harclerode', 18.0)
('john', 'smith', 32.0)
('jane', 'doe', 15.0)
```

To add column names to the cursor we can use the included row factory
for sqlite. The [python documentation](#) states that it should be more
performant than a custom implementation.

```
connection.row_factory = sqlite3.Row
for row in connection.execute('SELECT * FROM person;'):
    print(row.keys())
    print(tuple(row))
```



30

SQLite

```
[ 'first_name', 'last_name', 'age' ]
('tina', 'harclerode', 18.0)
[ 'first_name', 'last_name', 'age' ]
('john', 'smith', 32.0)
[ 'first_name', 'last_name', 'age' ]
('jane', 'doe', 15.0)
```



31

SQLite

```
try:
    with connection:
        connection.execute("""
            INSERT INTO person (first_name, last_name, age)
            VALUES (?, ?, ?)
            """, ('bob', 'smith', 54))
        connection.execute("this should fail")
except sqlite3.OperationalError:
    print('SQL statment failed with Operational error')

for row in connection.execute('SELECT * FROM person;'):
    print(tuple(row))
```



32

SQLite

```
(0, 'first_name', 'TEXT', 0, None, 0)
(1, 'last_name', 'TEXT', 0, None, 0)
(2, 'age', 'FLOAT', 0, None, 0)
(3, 'birthday', 'DATETIME', 0, None, 0)
(4, 'interests', 'JSON', 0, None, 0)
```

Next, we will insert interest for myself into the database.



33

SQLite

```
with connection:
    interests = {
        'books': ['C Programming Language', 'Daniels\' Running Formula'],
        'hobbies': ['running', 'programming']
    }
    connection.execute("""
    UPDATE person
    SET interests = ?
    WHERE first_name = ? AND last_name = ?
    """, (interests, 'tina', 'harclerode'))

for row in connection.execute('SELECT * FROM person;'):
    print(tuple(type(_) for _ in row))
    print(tuple(row))
```



34

SQLite

```
(<class 'str'>, <class 'str'>, <class 'float'>, <class 'NoneType'>, <class 'dict'>)
('tina', 'harclerode', 18.0, None, {'books': ['C Programming Language',
'Daniels' Running Formula'], 'hobbies': ['running', 'programming']})
(<class 'str'>, <class 'str'>, <class 'float'>, <class 'datetime.datetime'>,
<class 'NoneType'>)
('john', 'smith', 32.0, datetime.datetime(1977, 4, 23, 0, 0), None)
(<class 'str'>, <class 'str'>, <class 'float'>, <class 'NoneType'>, <class
'NoneType'>)
('jane', 'doe', 15.0, None, None)
```



35

SQLite

SQLite allow you to create functions and aggregate functions.

The python interface to SQLite provides a convenient method of creating functions that will be directly executed within SQLite. These functions will run directly inside of SQLite.

Create a simple json path function to return subpath of a json data:



36

SQLite

```
import re
sqlite3.enable_callback_tracebacks(True)

def json_path(json_path, blob):
    if blob is None: return None
    paths = [int(_) if re.match('\d+', _) else _ for _ in json_path.split('.')]
    path_value = json.loads(blob.decode())
    for path in paths:
        path_value = path_value[path]
    if isinstance(path_value, (int, float, str, bytes)):
        return path_value
    return (json.dumps(path_value)).encode()

with connection:
    connection.create_function("json_path", 2, json_path)
```



37

SQLite

Using sqlite custom functions can really help performance for queries that would normally return a large amount of data. This example can query what could be large json files.

If perform json_path on each value we reduce the amount of data the query returns. Since a json path can return another json object see SQLite CAST expression.

State the type of the return expression. In this case:

CAST json_path("books.1", interests) AS JSON.



38

SQLite

```
with connection:
    for row in connection.execute('SELECT json_path(?, interests) FROM
person', ("books.1",)):
        print(row)
```

```
("Daniels' Running Formula",)
(None,)
(None,)
```

OVERALL PYTHON EXPOSES A BEAUTIFUL INTERFACE TO SQLITE.



39

MODULE 04 HELPS

- <https://www.guru99.com/python-json.html>
- <https://www.tutorialspoint.com/numpy/index.htm>
- <https://www.pythonforthelab.com/blog/introduction-to-storing-data-in-files/>
- <https://www.pythonforthelab.com/blog/storing-binary-data-and-serializing/>
- <https://www.pythonforthelab.com/blog/storing-data-with-sqlite/>



40

Module 04 DISCUSSION FORUM

LEVEL OF ACHIEVEMENT				
Criteria	Exemplary	Proficient	Developing	Limited
Quality of Comments Total Points:	90 to 100 % Timely and appropriate comments as defined by the instructor. Thoughtful and reflective, responds respectfully to other students' remarks, provides questions and comments from the group.	75 to 89 % Volunteers comments, most are appropriate and reflect some thoughtfulness as defined by the instructor. Leads to other questions or remarks from students.	60 to 74 % Volunteers comments but lacks depth, may or may not lead to other questions from students. Off topic or irrelevant contributions.	0 to 59 % Did not participate.
Interaction with Course Materials Total Points:	90 to 100 % Clear reference to text being discussed and connects it to other text of relevance points from previous readings and discussions.	75 to 89 % Has done the reading with some thoroughness, may lack some detail or critical insight.	60 to 74 % Has done the reading, lacks thoroughness or understanding or insight.	0 to 59 % Did not participate.
Active Listening and Participation Total Points:	90 to 100 % Comments clearly demonstrate respect and attentiveness to others, creatively builds on others' comments to offer insights. Directly answers the question(s) asked AND provides additional insights. Exceptional level of interaction as defined by the instructor.	75 to 89 % Shows consistency in responding to the comments of others. Stay focused on the stream of discussion rather than own ideas. Directly answers the question(s) asked. Appropriate level of interaction as defined by the instructor.	60 to 74 % Does not stay focused on others' comments (too busy formulating next or lines continuity of discussion. Insistent in tracking discussion stream. Indirectly answers the assigned question(s). Poor level of interaction as defined by the instructor.	0 to 59 % Did not participate.



41

MODULE 04 DISCUSSION FORUM

You are working as an analytics developer and need to prepare the datasets to solve the business problem. Most of your datasets have some missing values in it.

For this discussion, describe and explain the following in your initial post. Your initial post should be a minimum of at least two fully-formed, well-thought-out scholarly paragraphs (blocks of code examples and graphics are considered additional information beyond the two required paragraphs but are encouraged when appropriate).

- Why is it required to change the data extraction process based on the business problem definition change?
- Give at least one example to that demonstrates the above requirement.



42

MODULE 04 DISCUSSION FORUM

For your reply, choose two other student responses and provide additional insights to each of them that add value to their posting. The reply should be at least one fully-formed, well-thought-out scholarly paragraph (blocks of code examples are considered additional information beyond the required paragraph but are encouraged when appropriate). A simple "I agree" type of post is unacceptable.

Due dates for your initial and response posts can be found by checking the Course Syllabus and Course Calendar.



43

MODULE 04 COURSE PROJECT: GRADING RUBRIC

Criteria	Points
A correct Python script is attached.	50
Output screenshots are attached.	50
Total	100



44

MODULE 04 COURSE PROJECT STORING OUTPUT

Please download the Noshowappointments.csv dataset from the Course Files page.

Then write a Python script that can perform following tasks.

- Step 1: Create two tables inside SQLite database
- Step 2: Store following outputs
 - Neighbourhood, count(No-Shows=Yes)
 - PatientID, count(No-Shows=Yes)

For your submission, include the following:

- Attach a Python script.
- An output screenshot.

Submit these files as a single zipped ".zip" file to the drop box below. Please check the Course Calendar for specific due dates.



45

MODULE 04 COURSE PROJECT

Note: For help with zipping or compressing your files, visit the [How do you zip files? Answers](#) page.

The name of the file should be your first initial and last name, followed by an underscore and the name of the assignment, and an underscore and the date. (Mac users, please remember to append the ".zip" extension to the filename.) An example is shown below:

Jstudent_exampleproblem_101504



46

MODULE 04 ASSIGNMENT: Grading Rubric

Criteria	Points
A correct Python script is attached.	50
Output screenshots are attached.	50
Total	100



47

MODULE 04 ASSIGNMENT STORING SCORECARDS

You are working as a software programmer for Rasmussen College. You need to write a prepare a database table to store the scorecard element of the different cohorts. Please refer to the Module_4_Assignment folder in the course zip file, located on the Course Files page.

You need to create a Python script that should perform following tasks.

- Create a SQLite table with following columns as described in the csv file and name it as Cohorts table
- Insert data into the SQLite table
- Store the data into the Pandas DataFrame
- Print the result using read_sql statement.



48

MODULE 04 ASSIGNMENT STORING SCORECARDS

For your submission, include the following:

- Attach a Python script.
- An output screenshot.

Submit these files as a single zipped ".zip" file to the drop box below.
Please check the Course Calendar for specific due dates.

Note: For help with zipping or compressing your files, visit the How do you zip files? Answers page.

The name of the file should be your first initial and last name, followed by an underscore and the name of the assignment, and an underscore and the date. (Mac users, please remember to append the ".zip" extension to the filename.) An example is shown below:

Jstudent_exampleproblem_101504