dataclasses-json

Typing meetup 2022-01-12

Overview

- About me
- Started writing this library in 2017/2018 when Python 3.7 (dataclasses) not yet released
- Focus on typing

Plan

- Tour
- Peek under the hood
- User feedback
- Future work

Tour

```python
import json
from dataclasses import dataclass, asdict
from dataclasses_json import dataclass_json
import timeit


@dataclass_json
@dataclass
class Person:
    name: str


person = Person(name='lidatong')
Person.schema().dumps(person)  # '{"name": "lidatong"}'

person_json = '{"name": "lidatong"}'
Person.schema().loads(person_json)  # Person(name='lidatong')
```

```
In [1]: from dataclasses import dataclass
   ...: from dataclasses_json import dataclass_json
   ...:
   ...:
   ...: @dataclass_json
   ...: @dataclass
   ...: class Person:
   ...:     name: str
   ...:

In [2]: Person.schema().loads('{"name": 42}')
```

```
    900         self.handle_error(exc, data, many=many, pa
--> 901         raise exc
    903 return result

ValidationError: {'name': ['Not a valid string.']}
```

Encode into a JSON array containing instances of my Data Class

```
people_json = [Person('lidatong')]
Person.schema().dumps(people_json, many=True)  # '[{"name": "lidatong"}]'
```

Decode a JSON array containing instances of my Data Class

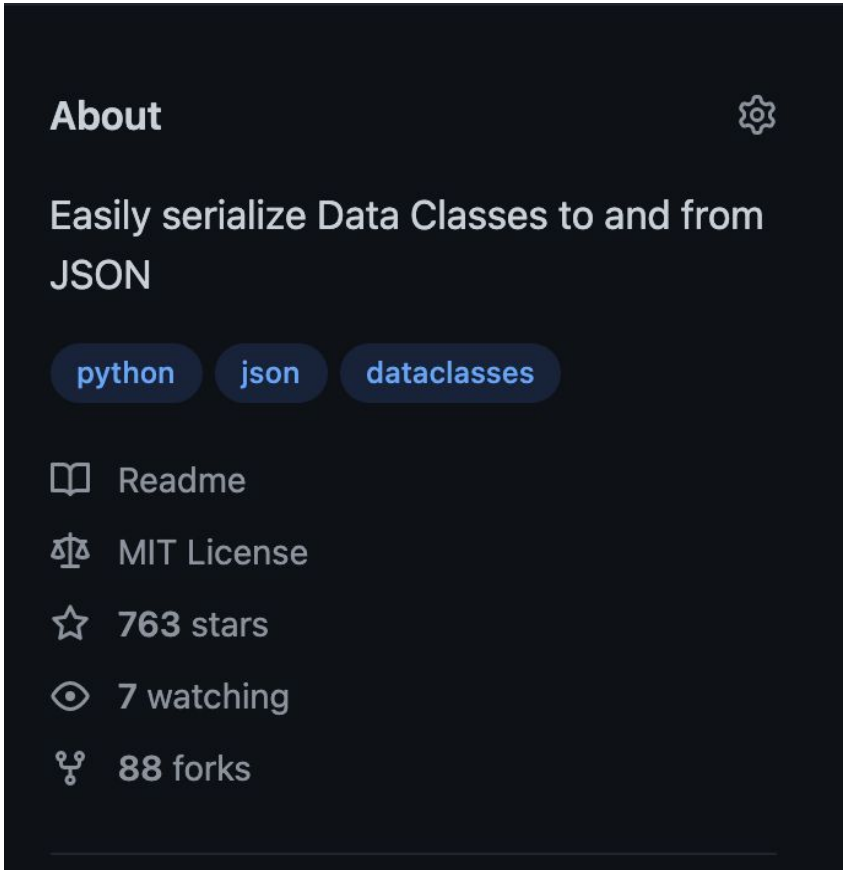
```
people_json = '[{"name": "lidatong"}]'
Person.schema().loads(people_json, many=True)  # [Person(name='lidatong')]
```

Encode as part of a larger JSON object containing my

```
import json

response_dict = {
    'response': {
        'person': Person('lidatong').to_dict()
    }
}

response_json = json.dumps(response_dict)
```

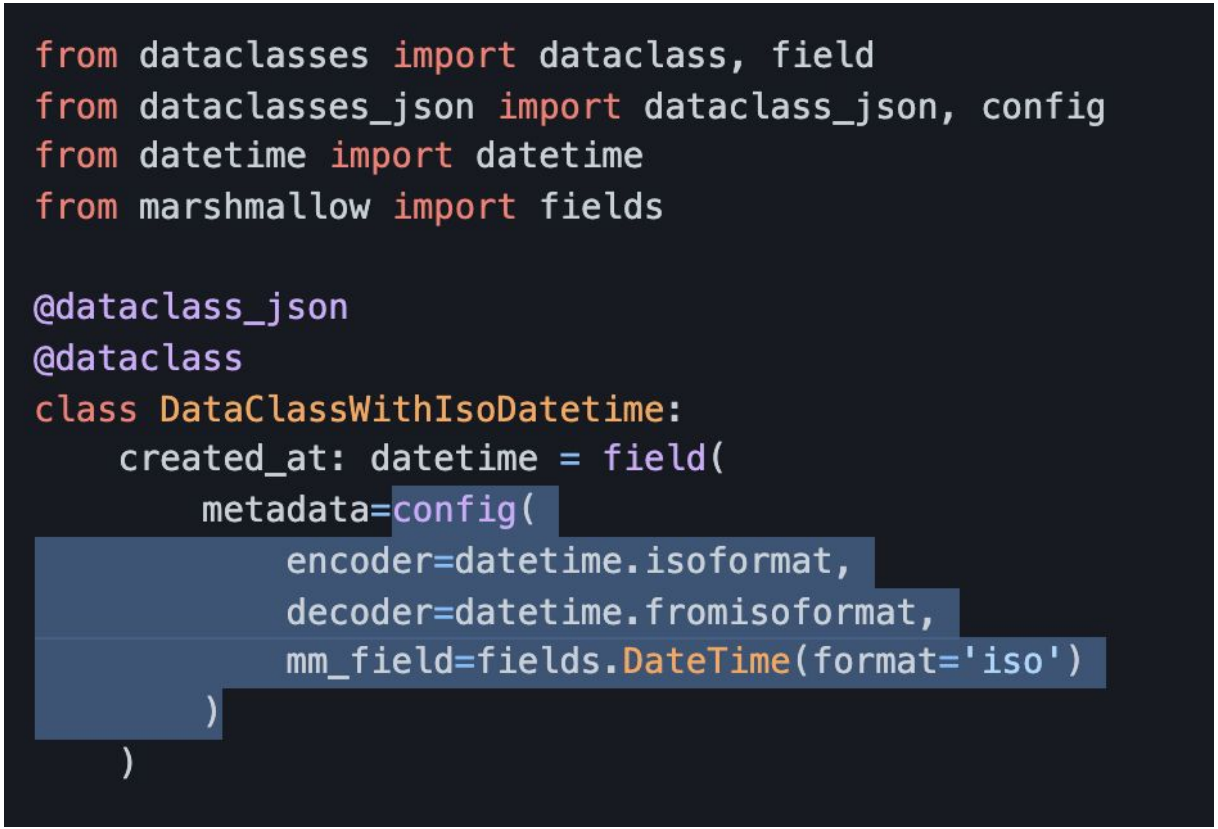
Decode as part of a larger JSON object containing my Data Class (e.g. an HTTP response)

```
import json

response_dict = json.loads('{"response": {"person": {"name": "lidatong"}}}')

person_dict = response_dict['response']

person = Person.from_dict(person_dict)
```


About
Easily serialize Data Classes to and from JSON
python
json
dataclasses
Readme
MIT License
763 stars
7 watching
88 forks

Highlights

API Extensibility



```
from dataclasses import dataclass, field
from dataclasses_json import dataclass_json, config
from datetime import datetime
from marshmallow import fields

@dataclass_json
@dataclass
class DataClassWithIsoDatetime:
    created_at: datetime = field(
        metadata=config(
            encoder=datetime.isoformat,
            decoder=datetime.fromisoformat,
            mm_field=fields.DateTime(format='iso')
        )
    )
```

Highlights

Error-handling

```
from dataclasses_json import Undefined

@dataclass_json(undefined=Undefined.RAISE)
@dataclass()
class ExactAPIDump:
    endpoint: str
    data: Dict[str, Any]

dump = ExactAPIDump.from_dict(dump_dict)  #
```

Peek under the hood

- Piggyback off **marshmallow** (schema library)
- **dataclasses** itself
- Private API of **typing**
  - __args__ to determine the type argument of the generic

User feedback

- (Caveat: this is my best guess interfacing with users on Github)
- Example use-cases:
  - Deserializing JSON payloads (web API)
  - Databases (Kafka, NoSQL etc.)
- First-class support one-off types
  - Extension mechanism
- Different API patterns
  - Often driven by user’s prior experience with another lib / another language…
  - My opinion: one way to do it
- **More typing support!!**

More typing support

- **[This code block]** does not type-check, can you investigate?
- Add support for this **[type]**.
- Can you make **[type error]** better?
- This **[typing feature]** came out, can you incorporate it?

User feedback

Support for pyspark.sql.Column in dataclasses-json #295

Earlier and More Helpful Type Errors #325

re-implementation of `_asdict` fails to parse `DataFrame` #311

support typing.Literal #317

Future work

- Typing support
  - Backstory – it wasn’t about **typing** at first!
- Deep-dive and understand the current typing landscape
- Many cool new developments and PEPs, and think about how I can bring them into dataclasses-json
- Many users have great suggestions and ideas, and need to condense into actionable development plan

Future work

- Performance and optimization
  - dataclasses-json is currently admittedly slow and not performant
  - Focus was on API usability and simplicity
  - Two dimensions of performance:
    - Library code
    - Underlying serde

```python
print(timeit.timeit(lambda: Person.schema().dumps(person), number=100) * 1000)
print(timeit.timeit(lambda: Person.schema().loads(person_json), number=100) * 1000)

schema = Person.schema()
print(timeit.timeit(lambda: schema.dumps(person), number=100) * 1000)
print(timeit.timeit(lambda: schema.loads(person_json), number=100) * 1000)

print(timeit.timeit(lambda: Person(**json.loads(person_json)), number=100) * 1000)
print(timeit.timeit(lambda: json.dumps(asdict(person)), number=100) * 1000)
```

```
32.88985299991509
38.2117480000943
1.1365269999714656
5.078514999922845
0.2943249999134423
0.7313020000765391
```

Thank you for listening