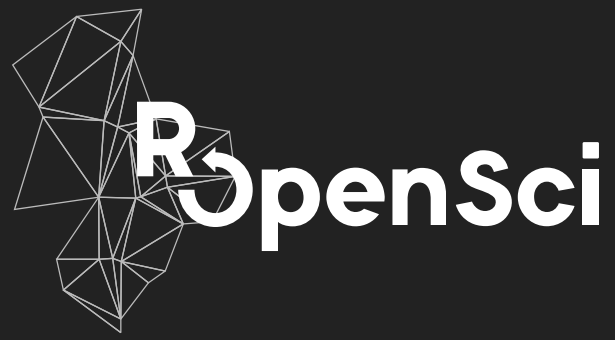
# staypuft: object validation and serialization

## & should this even be a package?

Scott Chamberlain (@sckottie)



# pain point: serialization

converting data in one format to another format

especially painful when complex

# other languages have good ideas

## marshmallow - a Python library

**marshmallow**

*A lightweight library for converting complex objects to and from simple Python datatypes.*

# An example with marshmallow

```
from datetime import date
from marshmallow import Schema, fields, pprint

class ArtistSchema(Schema):
    name = fields.Str()

class AlbumSchema(Schema):
    title = fields.Str()
    release_date = fields.Date()
    artist = fields.Nested(ArtistSchema())

bowie = dict(name='David Bowie')
album = dict(artist=bowie, title='Hunky Dory', release_date=date(1971, 12, 17))

schema = AlbumSchema()
result = schema.dump(album)
# { 'artist': {'name': 'David Bowie'},
#   'release_date': '1971-12-17',
#   'title': 'Hunky Dory'}

album = dict(artist=bowie, title='Hunky Dory', release_date="2020-04-14")
schema.dump(album)
# ValidationError: {'release_date': ['"2020-04-14" cannot be formatted as a date.']
```

# back to R

# similar art in R

- assertr (assertions for analysis pipeline)
- validate (very similar to assertr AFAICT)
- errorlocate (find errors in datasets)

- any others?

# staypuft

ropensci/staypuft

# An example with staypuft

```
library(staypuft)

MySchema <- Schema$new("MySchema",
  name = puft_fields$character(),
  title = puft_fields$character(),
  num = puft_fields$integer(),
  uuid = puft_fields$uuid(),
  foo = puft_fields$boolean()
)
x <- list(name = "Jane Doe", title = "Howdy doody", num = 5.5,
    uuid = "9a5f6bba-4101-48e9-a7e3-b5ac456a04b5",
    foo = TRUE)

# all good
MySchema$dump_json(x)
#> {"name":["Jane Doe"],"title":["Howdy doody"],"num":[5.5],
#>  "uuid":["foo-bar"],"foo":[true]}

# invalid uuid
x$uuid <- "foo-bar"
MySchema$load(data = x)
#> Error: ValidationError: Not a valid UUID.
```

# why?/use cases

- data validation: lots of potential users
- remote data sources can change: schemas help validate and catch changes
- use in scripts (most researchers): help raise issues with scripts as time goes on and data inputs change
- using R with plumbr or similar: convert data to serve to API or consume from API request bodies

# features

- user created schemas
- seralize among object types (R6/data.frame/JSON)
- make dealing with nested data easier
- specify required fields
- specify default fields
- specify order of output fields

# To do

- Nested data - this could be huge, reducing pain in flattening out nested data
- Lots more 'field' types: url, email, (domain specific types)
- Probably add an easier to use interface, less R6'y

wait ...
should this even be a package though?

# When I should not make software

- the pkg doesn't solve actual use cases
- there's significant overlap with existing solutions
  - and maintainers are responsive
- I can't envision maintaining the package for the long run
- I lack sufficient knowledge in the topic area
- there's higher priority/lowering hanging fruit

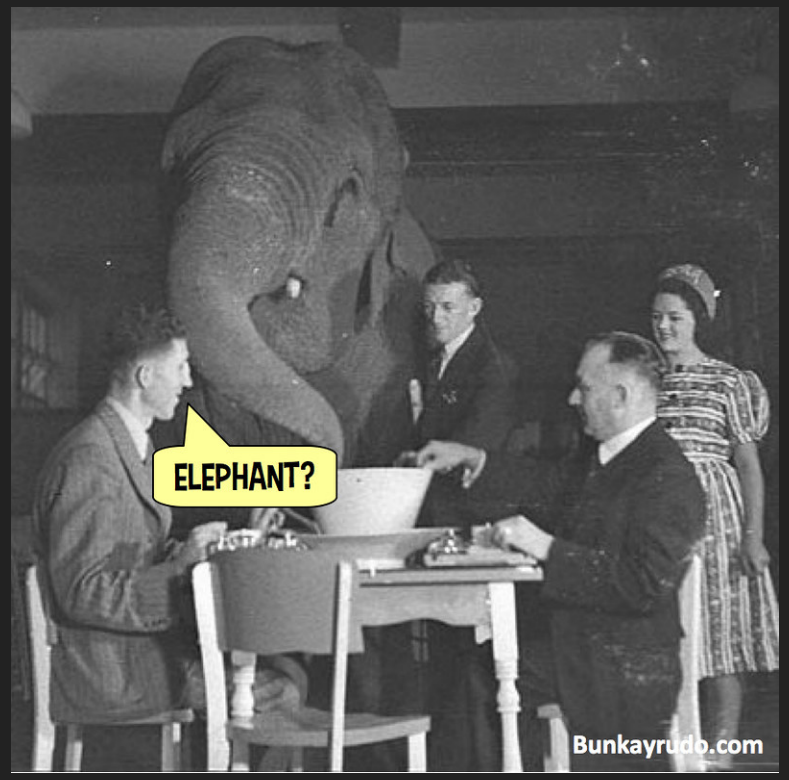
# not arguing against sillyness

**cowsay: Messages, Warnings, Strings with Ascii Animals**

Allows printing of character strings as messages/warnings/etc. with ASCII animals, including cats, cows, frogs, chickens, ghosts, and more.

| | |
|---|---|
| Version: | 0.8.0 |
| Imports: | crayon, fortunes, rmsfact |
| Suggests: | curl, jsonlite, knitr, multicolor, rmarkdown, testthat |
| Published: | 2020-02-06 |
| Author: | Scott Chamberlain [aut, cre], Amanda Dobbyn [aut], Tyler Rinker [ctb], Thomas Leeper [ctb], Noam Ross [ctb], Rich FitzJohn [ctb], Carson Sievert [ctb], Kiyoko Gotanda [ctb], Andy Teucher [ctb], Karl Broman [ctb], Franz-Sebastian Krah [ctb], Lucy D'Agostino McGowan [ctb], Guangchuang Yu [ctb], Philipp Boersch-Supan [ctb], Andreas Brandmaier [ctb], Marion Louveaux [ctb] |

# elephant in the room ...



# S4 e.g.

```
setClass("BMI", representation(weight="numeric", size="numeric"))
new("BMI", weight="Hello", size=1.84)
#> Error in validObject(.Object) :
#>   invalid class “BMI” object: invalid object for slot "weight"
#>   in class "BMI": got class "character",
#>   should be or extend class "numeric"
setValidity # check validity
```

# Use cases

For `staypuft`, likely many users

Everyone deals with objects in R

# higher priority/lower hanging fruit

- I've got many other packages
- Many of which have many users
- What if new package has a huge impact though?
  - How would I know?

# the end