ada-benchmark-notebook

Jane Doe

3/7/25

Table of contents

Preface

This is a Quarto book.

To learn more about Quarto books visit https://quarto.org/docs/books.

1 Quarto Computations

```
import numpy as np
a = np.arange(15).reshape(3, 5)
a
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

1.1 Matplotlib

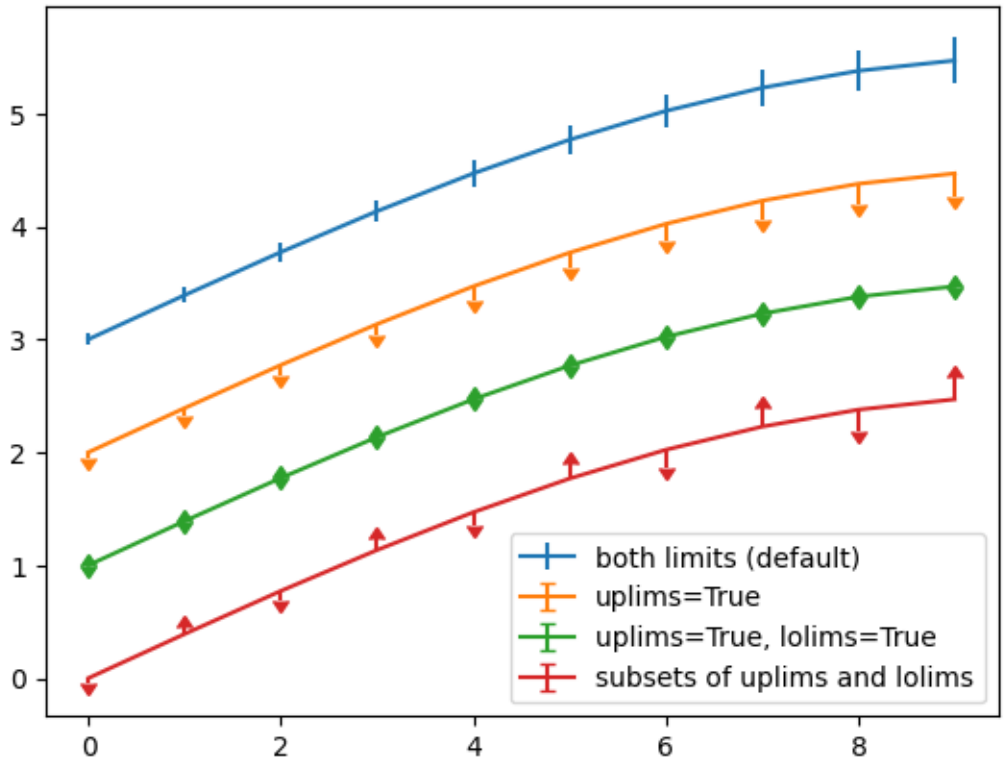
```
import matplotlib.pyplot as plt

fig = plt.figure()
x = np.arange(10)
y = 2.5 * np.sin(x / 20 * np.pi)
yerr = np.linspace(0.05, 0.2, 10)

plt.errorbar(x, y + 3, yerr=yerr, label='both limits (default)')
plt.errorbar(x, y + 2, yerr=yerr, uplims=True, label='uplims=True')
plt.errorbar(x, y + 1, yerr=yerr, uplims=True, lolims=True,
             label='uplims=True, lolims=True')

upperlimits = [True, False] * 5
lowerlimits = [False, True] * 5
plt.errorbar(x, y, yerr=yerr, uplims=upperlimits, lolims=lowerlimits,
             label='subsets of uplims and lolims')

plt.legend(loc='lower right')
plt.show(fig)
```



1.2 Plotly

```
import plotly.express as px
import plotly.io as pio
gapminder = px.data.gapminder()
gapminder2007 = gapminder.query("year == 2007")
fig = px.scatter(gapminder2007,
                 x="gdpPercap", y="lifeExp", color="continent",
                 size="pop", size_max=60,
                 hover_name="country")
fig.show()
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json
```

2 Query ORCID for works authored by a person

Taken from Project TAPIR's repository of Jupyter Notebooks: https://github.com/Project-TAPIR/pidgraph-notebooks

This notebook queries the ORCID Public API to retrieve works listed in a person's ORCID record. It takes an ORCID URL or iD as input to retrieve the ORCID record of a person and the works listed on it. From the resulting list of works we output all DOIs.

```
# Prerequisites:
import requests                          # dependency to make HTTP calls
from benedict import benedict            # dependency for dealing with json
```

The input for this notebook is an ORCID URL or iD, e.g. '`https://orcid.org/0000-0003-2499-7741`' or '`0000-0003-2499-7741`'.

```
# input parameter
example_orcid="https://orcid.org/0000-0002-2437-589X"
```

We use it to query ORCID's Public API for the person's metadata and all works connected to them.

```
# URL for ORCID API
ORCID_RECORD_API = "https://pub.orcid.org/v3.0/"

# query ORCID for an ORCID record
def query_orcid_for_record(orcid_id):

    response = requests.get(url=requests.utils.requote_uri(ORCID_RECORD_API + orcid_id),
                            headers={'Accept': 'application/json'})
    response.raise_for_status()
    result=response.json()
    return result
```

```
#-- example execution
orcid_id=example_orcid.replace("https://orcid.org/", "")
orcid_record=query_orcid_for_record(orcid_id)
# uncomment next lines to see complete metadata for given ORCID
#import pprint
#pprint.pprint(orcid_record)
```

From the complete ORCID metadata we extract the works section and print out title and DOI of each first `work-summary` (the first item in a personal information section has the highest display index).

Note: works that do not have a DOI assigned, will not be printed.

```
# extract works section from ORCID profile
def extract_works_section(orcid_record):
    orcid_dict=benedict.from_json(orcid_record)
    works=orcid_dict.get('activities-summary.works.group') or []
    return works

# for each work in the work section: extract title and DOI
def extract_doi(work):
    work_dict=benedict.from_json(work)
    title=work_dict.get('work-summary[0].title.title.value')
    dois= [doi['external-id-value'] for doi in work_dict.get('work-summary[0].external-ids
    # if there is a DOI assigned to the work, the list of dois is not empty and we can ext
    doi=dois[0] if dois else None
    return doi, title


# ---- example execution
works=extract_works_section(orcid_record)
for work in works:
    doi,title = extract_doi(work)
    if doi:
        print(f"{doi}, {title}")
```

```
10.21428/785a6451.af466093, What is computational publishing?
10.21428/785a6451.1792b84f, Books Contain Multitudes
10.21428/785a6451.0ed93b68, Part 1: Mapping & Situating Experimental Books
10.21428/785a6451.cd58a48e, Part 2: A Typology of Experimental Books
10.21428/785a6451.c4d3ffa1, Introducing COPIM's new website
10.17613/d4rs-rx03, Posthumanism in Outer Wilds
```

10.17613/5SJE-W447, New Sincerity, the Weird, and the post-ironic turn in contemporary indie
10.20944/preprints202001.0240.v1, Open or Ajar? Openness within the Neoliberal Academy

3 Embedded video

The below Python code experiments with retrieving video data via iframe embedding.

```
from IPython.display import HTML
HTML("""
<iframe width="560" height="315" scrolling="no" src="https://av.tib.eu/player/56162" frame
""")
```

```
<IPython.core.display.HTML object>
```

4 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

5 Summary

In summary, this book has no content whatsoever.

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.