Doctor

Elliott Phillips

/ellsphillips/doctor

24/03/2022

Head's up!I'm a full-time senior Data Scientist, currently on secondment and juggling projects as I go...

But more on that later...

Introduction

- About
- Users
- Design

About – What is Doctor?

An automated documentation assistant built in Python and TEX for procedural, data-driven reporting.

Doctor provides services to simplify the reporting of data-oriented, beautiful, lightweight documents.

Transparent and opinionated, without the WYSIWYG faff.

About – Journey

Personal side project

- LaTeX University requirement
- Self-taught Python at ONS
- Wanted to inject flair whilst automating corporate documents

Business-critical application

- Lead project to deliver real-time financial estimates
- Complex data to visualise and communicate
- Templated cohesion between report releases

Internal collaborative platform

- SCS requested S&T to National Statistician
- Delivered demoes, assembled cross-ONS development team
- Coach colleagues in software design and effective versioning

About – Journey

Personal side project

- LaTeX University requirement
- Self-taught Python at ONS
- Wanted to inject flair whilst automating corporate documents

Business-critical application

- Lead project to deliver real-time financial estimates
- Complex data to visualise and communicate
- Templated cohesion between report releases

Internal collaborative platform

- SCS requested S&T to National Statistician
- Delivered demoes, assembled cross-ONS development team
- Coach colleagues in software design and effective versioning

About – Journey

Personal side project

- LaTeX University requirement
- Self-taught Python at ONS
- Wanted to inject flair whilst automating corporate documents

Business-critical application

- Lead project to deliver real-time financial estimates
- Complex data to visualise and communicate
- Templated cohesion between report releases

Internal collaborative platform

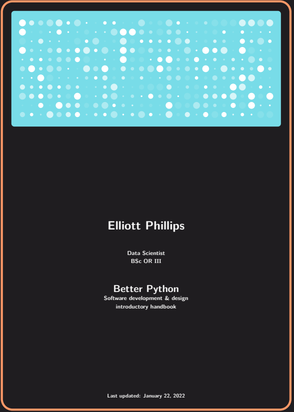
- SCS requested S&T to National Statistician
- Delivered demoes, assembled cross-ONS development team
- Coach colleagues in software design and effective versioning

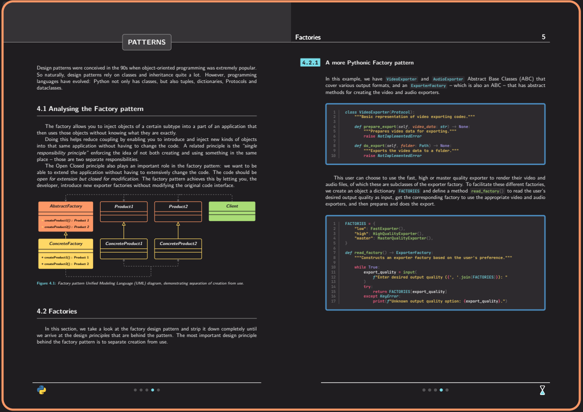
As technical project lead, requirement to transpose regular income data stream into a monthly insights report on businesses' resilience in response to COVID-19, delivering to CO...

Users – Corporate reports

As technical project lead, requirement to transpose regular income data stream into a monthly insights report on businesses' resilience in response to COVID-19, delivering to CO...

Automate the boring stuff!





PATTERNS

Design patterns were conceived in the 90s when object-oriented programming was extremely popular. So naturally, design patterns rely on classes and inheritance quite a lot. However, programming languages have evolved: Python not only has classes, but also tuples, dictionaries, Protocols and dataclasses.

4.1 Analysing the Factory pattern

The factory allows you to inject objects of a certain subtype into a part of an application that then uses those objects without knowing what they are exactly.

Doing this helps reduce coupling by enabling you to introduce and inject new kinds of objects into that same application without having to change the code. A related principle is the *"single responsibility principle"* enforcing the idea of not both creating and using something in the same place – those are two separate responsibilities.

The Open Closed principle also plays an important role in the factory pattern: we want to be able to extend the application without having to extensively change the code. The code should be *open for extension but closed for modification*. The factory pattern achieves this by letting you, the developer, introduce new exporter factories without modifying the original code interface.

Figure 4.1: *Factory pattern Unified Modeling Language (UML) diagram, demonstrating separation of creation from use.*

4.2 Factories

In this section, we take a look at the factory design pattern and strip it down completely until we arrive at the design *principles* that are behind the pattern. The most important design principle behind the factory pattern is to separate creation from use.

4.2.1 A more Pythonic Factory pattern

In this example, we have `VideoExporter` and `AudioExporter` Abstract Base Classes (ABC) that cover various output formats, and an `ExporterFactory` – which is also an ABC – that has abstract methods for creating the video and audio exporters.

```python
class VideoExporter(Protocol):
    """Basic representation of video exporting codec."""

    def prepare_export(self, video_data: str) -> None:
        """Prepares video data for exporting."""
        raise NotImplementedError

    def do_export(self, folder: Path) -> None:
        """Exports the video data to a folder."""
        raise NotImplementedError
```

This user can choose to use the fast, high or master quality exporter to render their video and audio files, of which these are subclasses of the exporter factory. To facilitate these different factories, we create an object a dictionary `FACTORIES` and define a method `read_factory()` to read the user's desired output quality as input, get the corresponding factory to use the appropriate video and audio exporters, and then prepares and does the export.

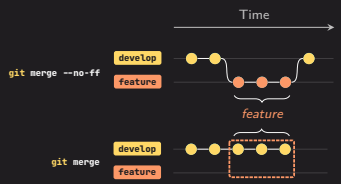
```python
FACTORIES = {
    "low": FastExporter(),
    "high": HighQualityExporter(),
    "master": MasterQualityExporter(),
}

def read_factory() -> ExporterFactory:
    """Constructs an exporter factory based on the user's preference."""

    while True:
        export_quality = input(
            f"Enter desired output quality ({', '.join(FACTORIES)}): "
        )
        try:
            return FACTORIES[export_quality]
        except KeyError:
            print(f"Unknown output quality option: {export_quality}.")
```

Model – Feature branches

16/21




```
$ git clone https://github.com/ellsphillips/git-book git-good
```

Examples

- Table
- Plot

Table – Container style options

C_1	C_2	C_3	C_4
a	b	c	d
e	f	g	h
i	j	k	l

C_1	C_2	C_3	C_4
a	b	c	d
e	f	g	h
i	j	k	l

Table – Flexibility

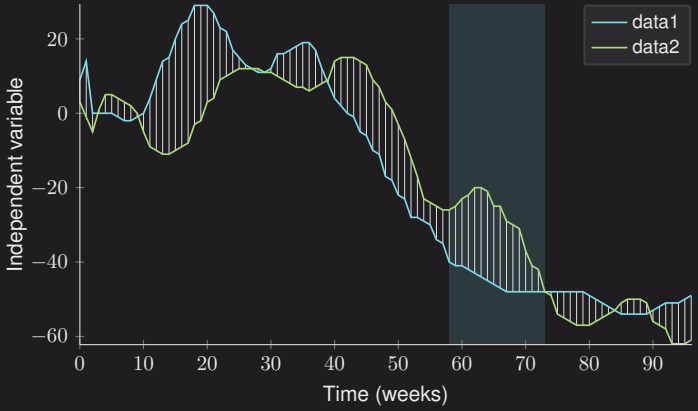
C_1	C_2	C_3	C_4
1.00	2.00	3.00	4.00
1.01	2.01	3.01	4.01
1.02	2.02	3.02	4.02
1.03	2.03	3.03	4.03
1.04	2.04	3.04	4.04
⋮	⋮	⋮	⋮

<<page break>>

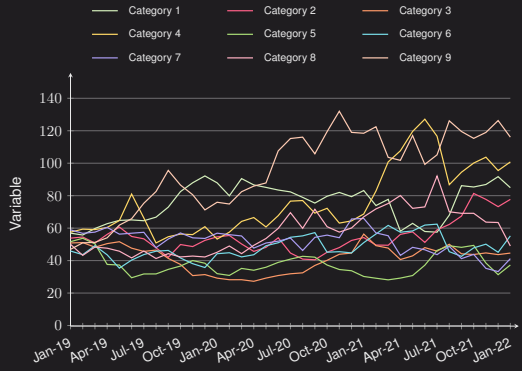
<<continue table>>

⋮	⋮	⋮	⋮
1.97	2.97	3.97	4.97
1.98	2.98	3.98	4.98
1.99	2.99	3.99	4.99

Plot – line

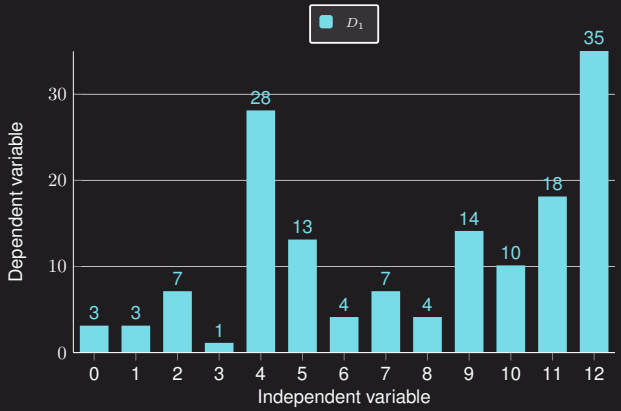

data1
data2
Independent variable
Time (weeks)
20
0
−20
−40
−60
0
10
20
30
40
50
60
70
80
90

Plot – line

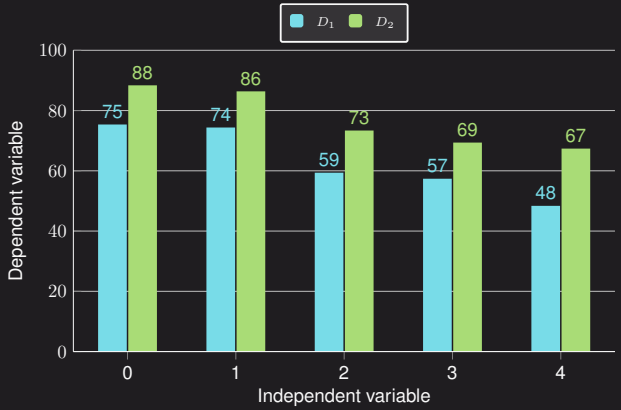


Source: company

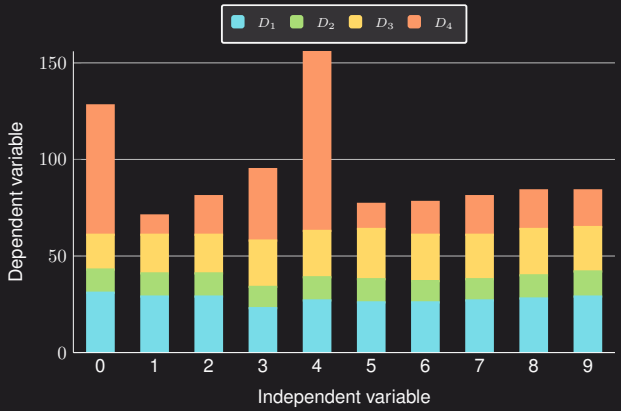
Plot – bar


D_1
Dependent variable
Independent variable
0
10
20
30
0 1 2 3 4 5 6 7 8 9 10 11 12
3 3 7 1 28 13 4 7 4 14 10 18 35

Plot – bar


D_1
D_2
100
80
60
40
20
0
Dependent variable
75
88
74
86
59
73
57
69
48
67
0
1
2
3
4
Independent variable

Plot – bar


D_1
D_2
D_3
D_4
150
100
50
0
Dependent variable
0
1
2
3
4
5
6
7
8
9
Independent variable

Usage

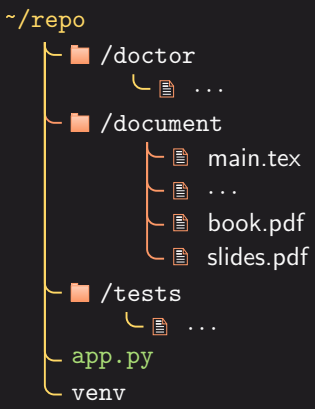
- Installation
- API

Installation – Project


```python
import doctor as dr


def main() -> None:
    ...


if __name__ == "__main__":
    main()
```


~/repo
/doctor
...
/document
main.tex
...
book.pdf
slides.pdf
/tests
...
app.py
venv

API – Tables


```python
tabular = dr.table(
    dr.data.table(
        [
            dr.data.numeric.rand(10),
            dr.data.text.lorem(10),
        ]
    ),
    columns=["numbers", "words"]
)
```

API – Tables


```
$  \begin{doctor-table}[%
$      columns={C_1, C_2, C_3, C_4}%
$  ]
$      Lorem        & Lorem     & Lorem   & Lorem     \\
$      ipsum        & ipsum     & ipsum   & ipsum     \\
$      sit          & rhoncus   & vel     & convallis \\
$      ut           & est       & ac      & mauris    \\
$      Nam          & id        & Morbi   & vitae     \\
$      laoreet      & egestas   & tellus. & eu        \\
$      sollicitudin & nisl      & sit     & eleifend  \\
$      Donec        & elementum & dui.    & amet      \\
$      sed          & conubia   & id      & amet      \\
$      sem          & odio.     & nisl    & in        \\
$  \end{doctor-table}
```

API – Plot


```
figure = dr.plot(
    "line",
    data={
        "timeseries_2020": dr.data.series.brownian(),
        "timeseries_2021": dr.data.series.brownian(),
        "timeseries_2022": dr.data.series.brownian(),
    },
    options={
        "plot type": "ybar",
        "data source": "src/plots/example.dat",
        "caption": "Demonstration of the doctor-plot env",
        "label": "example-plot",
    },
)
```

API – Plot


```
$  \begin{doctor-plot}[%
$      plot type={ybar},
$      data source={src/plots/example.dat},
$      caption={Demonstration of the doctor-plot environment},
$      label={example-plot}%
$  ]
$      \addplot+[%
$          ons-pink,
$          thick,
$          mark=none%
$      ] table[x=time, y=some_data]%
$      {src/graphs/timeseries.dat};
$  \end{doctor-plot}
```

Document class API

Next steps

- Package

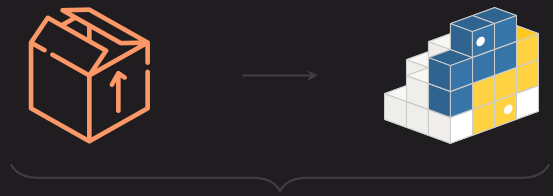
- Refine and publish existing PoC test library
- Deliver testing fundamentals workshop
- Build robust testing suite for Doctor

1. Charting library (Scatter, Pie, ...)
2. Choropleth support
3. Rendering local code with syntax highting

⋮

n. Any ideas?

Package – Deployment

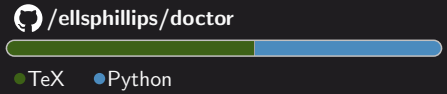


```
$ pip install doctor
```

Contribution

- GitHub

Interested in contributing? Doctor is developed open source! Get it touch via email or create a pull request



You're welcome to retain a copy and share this material with anyone who may benefit.

Please ★ this repository if you have found this material useful and to follow its development!

Doctor

Elliott Phillips

/ellsphillips/doctor

24/03/2022