Example of Converted Jupyter Notebook

Formatting Code Cells

Author: Chris Sewell
chrisj_sewell@hotmail.com

Supervisors:
First Supervisor
Second Supervisor

Converted using IPyPublish
(‘latex_ipypublish_all.exec’).

Institution1
Institution2

20th February 2019
1 Writing Code and Formatting Output

IPyPublish utilises metadata to mark-up the notebook with information on how output should be represented in the converted notebook, as shown in fig. 1.1.

```python
import matplotlib.pyplot as plt
import numpy as np
plt.plot(np.sin(np.linspace(0, 6)))
plt.show()
```

**Figure 1.1:** This is a Matplotlib figure, with a caption, a label and a set width

**seealso**

??, for a full description and list of ipypublish metadata

1.1 Converting Notebooks to Pure Python

To write code, we can work in the conventional Jupyter Notebook environment, or we can use jupyter, to convert between a notebook and the pure python percent format:

```bash
$ jupyter --to py:percent notebook.ipynb
$ jupyter --to notebook notebook.py # overwrite notebook.ipynb
$ jupyter --to notebook --update notebook.py # update notebook.ipynb
```

This will produce a standard python file, with commented notebook level metadata commented at the top (in YAML format), and each cell beginning with `# %%` (known as the percent format):

The percent format can be utilised in IDEs, such as Spyder, Atom, PyCharm, and VS Code, to run individual cells:

**important**

To preserve ipypublish notebook metadata, you must add: "jupyter": {"metadata_filter": {"notebook": "ipub"}} to your notebooks metadata before conversion.

**seealso**

??

Using YAML metadata blocks in Pandoc.
1.2 NB Setup Helper Functions

offers a number of useful functions, to setup common packages (matplotlib, pandas, etc) for outputting content in high quality formats.

```python
from ipypublish import nb_setup
```

```
note

ipypublish.scripts.ipynb_latex_setup is deprecated in v0.9
```

1.3 Text Output

```python
print("\nThis is some printed text,\nwith a nicely formatted output.\n")
```

This is some printed text, with a nicely formatted output.

1.4 Images (with PIL)

```python
import os
from ipypublish.tests import TEST_FILES_DIR
example_pic = os.path.join(TEST_FILES_DIR, 'example.jpg')

nb_setup.images_hconcat([example_pic, example_pic],
width=600, gap=10)
```
Figure 1.3: Horizontally aligned images.

```python
nb_setup.images_vconcat([example_pic, example_pic],
    height=400, gap=10)
```

Figure 1.4: Vertically aligned images.
nb_setup.images_gridconcat([example_pic, example_pic],
    height=300, vgap=10, hgap=20)

Figure 1.5: Images aligned in a grid.

1.5 Plots (with Matplotlib)

A matplotlib figure (fig. 1.6), and its code (code 1.1).

Code 1.1: The plotting code for a matplotlib figure (fig. 1.6).

```python
plt = nb_setup.setup_matplotlib(output=('pdf', 'svg'))
plt.scatter(np.random.rand(10), np.random.rand(10),
    label='data label')
plt.ylabel(r'$a y label with latex $\alpha$')
plt.legend();
```
Figure 1.6: A matplotlib figure

If outputting the Matplotlib figures in a PDF format. See usetex tutorial, and Stackoverflow question.

1.6 Tables (with pandas)

A pandas table (table 1.1), and its code (code 1.2).

Code 1.2: The plotting code for a pandas Dataframe table (table 1.1).

```python
1 pd = nb_setup.setup_pandas(escape_latex=False)
2 df = pd.DataFrame(np.random.rand(3,4), columns=['a', 'b', 'c', 'd'])
3 df.a = ['$\delta$', 'x', 'y']
4 df.b = ['l', 'm', 'n']
5 df.set_index(['a', 'b'])
6 df.round(3)
```

Table 1.1: An example of a table created with a pandas dataframe.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>δ</td>
<td>l</td>
<td>0.407</td>
<td>0.343</td>
</tr>
<tr>
<td>1</td>
<td>x</td>
<td>m</td>
<td>0.137</td>
<td>0.628</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td>n</td>
<td>0.657</td>
<td>0.154</td>
</tr>
</tbody>
</table>

If using escape_latex=False, then PDF conversion will throw an error if there are e.g. _’s in your
1.7 Equations (with ipython or sympy)

An ipython and sympy equation (1.1) and (1.2).

```python
from IPython.display import Latex
Latex('$$ a = b+c $$')
```

\[ a = b + c \] (1.1)

Code 1.3: The plotting code for a sympy equation (1.2).

```python
sym = nb_setup.setup_sympy()
f = sym.Function('f')
y,n = sym.symbols('y \alpha')
f = y(n)-2*y(n-1)/sym.pi-5*y(n-2)
sym.rsolve(f,y(n),[1,4])
```

\[
(\sqrt{5}i)^{\alpha} \left( \frac{1}{2} - \frac{2i}{5} \sqrt{5} \right) + (-\sqrt{5}i)^{\alpha} \left( \frac{1}{2} + \frac{2i}{5} \sqrt{5} \right) \] (1.2)

1.8 Object Output Formats

The format of the Jupyter Notebook file allows for the storage of a single output in multiple formats. This is taken advantage of by packages such as matplotlib and pandas, etc to store a figure/table in both latex and html formats, which can then be selected by ipypublish based on the document type required.

Sometimes a user may wish to have greater control over the output format and/or which output types are to be stored. It is possible to achieve this via the Jupyter display function. For example, if we wanted to display a pandas.DataFrame table without the index column, such that it can be output to both a pdf and html document:

```python
from IPython.display import display
df = pd.DataFrame(np.random.random((3, 3)))
latex = df.to_latex(index=False)
html = df.to_html(index=False)
display({'text/latex': latex,
 'text/html': html}, raw=True)
```

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.723444</td>
<td>0.834081</td>
<td>0.973820</td>
</tr>
<tr>
<td></td>
<td>0.284984</td>
<td>0.830689</td>
<td>0.049264</td>
</tr>
<tr>
<td></td>
<td>0.045792</td>
<td>0.303503</td>
<td>0.707279</td>
</tr>
</tbody>
</table>

If you wish to create your own object with multiple output formats, you should create a class with multiple _repr_*_() methods:
```python
class MyObject(object):
    def __init__(self, text):
        self.text = text

    def _repr_latex_(self):
        return r'\textbf{LaTex: }' + self.text + '"

    def _repr_html_(self):
        return '<b>HTML: ' + self.text + '</b>

MyObject('hallo')
```

**seealso**

??

IPython Rich Display

### 1.9 Multiple Outputs from a Single Code Cell

Similarly, with the Jupyter display functionality, you can control the output metadata for multiple outputs in a single code cell:

```python
from IPython.display import display
from IPython.display import display_latex
from IPython.display import display_markdown

x = np.linspace(0, 3.42)

for i in range(1,3):
    display_markdown(
        '### Code Created Heading \{0\}'.format(i), raw=True)

    fig, ax = plt.subplots()
    ax.plot(x, np.sin(x*i))
    metadata={'ipub': {
        'figure': {  
            'caption': 'Code Created Heading \{0\}'.format(i)}}}
    display(fig, metadata=metadata)
plt.close()
```
1.9.1 Code Created Heading 1

*Figure 1.7*: Code Created Heading 1
1.9.2 Code Created Heading 2

Figure 1.8: Code Created Heading 2