easytex

May 13, 2019

Contents

List of Figures

List of Tables

1 Introduction

The nypd_reports library is intended to serve as easy-to-use python-to-latex API interface. Rather then having to know exactly what latex commands to use, the intention is for nypd_reports to do the heavy lifting for you. The nypd_reports library is currently organzied into 8 classes:tex

1. Preamble
2. Document
3. Text
4. Table
5. Figure
6. Section
7. Environment
8. PageStyle

Each of these classes, and their uses are documented in the following sections.

The nypd_reports library can be loaded simply by running:

```
from nypd_reports import *
```

2 Preamble

The preamble section contains all the necessary setup information for latex to render a document. The Document class (covered later) requires a Preamble object upon declaration, which means making a preamble should be the first step when creating a new document. By default, the Preamble() class loads a default preamble. The only values a user is required to pass pertain to document meta-inforamation including: 1) Title, 2) Author, 3) Date, and 4) font. Note that the font passed is document-wide.

The Preamble class accepts the following arguments:

1. tex: str
   (a) Can pass a custom preamble here.
2. font: str
   (a) Name of font to use with the document.
3. title: str
   (a) Title of the report.
4. author: str
   (a) Author(s) of the report.
5. date: str
   (a) String representing the date to appear on the document.

2.1 Example Usage

```
preamble = Preamble(author='', date='\today', title='easytex', font='Lato')
```

3 Document

The Document class represents the full latex document. A Document object requires a preamble object upon initialization. By default, a new latex document always includes the provided title, a table of contents, a list of figures and a list tables. These can all be disabled when creating a new Document object.

The document class, like all provided classes as part of the nypd_reports library, includes a .add() function. This function represents the intended way to combine the different report parts (such as text, tables and figures) together into a cohesive document. A typical workflow follows the pattern: 1) Create Preamble, 2) Create Document (using Preamble), 3) Create Content, 4) Add content to Document, 5) Repeat 3 & 4 untill finished.

The Document class accepts the following arguments:

1. preamble: Preamble
   (a) A Preamble object.
2. include_title: bool
   (a) Name of font to use with the document.
3. table_of_contents: bool
   (a) True: Include a table of contents.
   (b) False: Do not include a table of contents.
4. list_of_figures: bool
   (a) True: Include a list of figures.
   (b) False: Do not include a list of figures.
5. list_of_tables: bools
   (a) True: Include a list of tables.
   (b) False: Do not include a list of tables.

3.1 Important Methods

Some important methods belonging to the Document class include:

1. add_clearpage(self)
   (a) Adds a clearpage command at the current document location.
2. export_tex(self, file)
   (a) Exports the documents tex commands to file.
3. render_report(self, destination)
   (a) Calls export_tex and then compiles the tex file using xelatex.
4. print_map(self)
   (a) Prints an outline view of the current document.

3.2 Example Usage

```
Example creating a document, adding a section, exporting the tex, and then rendering the pdf:
```

```
# Create a new document:
document = Document(
    preamble=preamble,
    include_title=True,
    table_of_contents=True,
    list_of_figures=True,
    list_of_tables=True
)

# Add a Section object named 'section':
document.add(section)

# Render the document:
document.render_report('rendered_report_name.pdf')

# Note: The rendered report will be saved under 'rendered_report_name.pdf'
```

4 Text

This class represents a latex text.

The Text class accepts the following arguments:

1. text: str / dict
   (a) Passed text will be treated as plain text for a latex document, or as a list of a dictionry is passed.
   (b) When passing a dictionary, the keys should be the list text, and the values correspond to the list level.
   (c) Example: dict('here is a list entry': 1)
2. verbatim: bool
   (a) If true, inserts text verbatim (as-is).
   (b) Useful for codeblocks.

4.1 Important Methods

Some important methods belonging to the Text class include:

1. add_list(self, list_items, bullets=False)
   (a) Creates a list from list_items.
   (b) If bullets=False, then bullets will be numeric.
2. add_bold(self, text)
   (a) Inserts bolded text.

4.2 Example Usage

Creating a list:

```
# Create list dictionary
# The numeric corresponds to the list level; '1' is the top level
# '2' is a sub bullet, and so on.

dict_ = {
    'add_list(self, list_items, bullets=False)': 1,
    'Creates a list from list_items.': 2,
    'If bullets=False, then bullets will be numeric.': 2,
    'add_verbatim(self, text)': 1,
    'Inserts text verbatim or as-is.': 2,
    'Useful for codeblocks.': 2,
    'add_bold(self, text)': 1,
    'Inserts bolded text.': 2
}

# Create a latex list using the Text() class from the above dict:
list_ = Text(dict_)
```

5 Table

This class represents a latex table. A Table object can be initialized in a number of ways, but by far the easiest is to pass a pandas DataFrame. While pandas includes a ~.to_latex() function, this can be severly limiting if one wants to change anything having to do with a particular table's style. The Table class provides a number of style arguments, including: row-lines, row highlighting, longtables and landscape tables, and zebra stripping.

The Table class is configured to create a minimalist table by default.

The Table class accepts the following arguments:

1. table_type: str
   (a) Must be 'table', 'tabular', 'longtable', or 'sidewaystable'
   (b) Designates the type of tabular environment used.
2. label: str
   (a) A string represnting the table's cross reference label
3. data: pd.DataFrame or list
   (a) A pandas dataframe or list object to convert into a latex table.
4. cols: list
   (a) List of column names. Used to generate an appropriately sized table.
5. caption: str
   (a) A string representing the table's caption.
6. captionof: str
   (a) A string representing the table's caption. To be used outside of a float environment.
7. empty_label: bool
   (a) A flag representing whether to use a blank caption.
8. alignment: str
   (a) A string representing a latex tabular alignment command.
   (b) Used to overwrite default alignment.
9. zebra: bool
   (a) Boolean representing whether the table should be zebra stripped or not.
10. row_colors: dict
11. Dictionary mapping row indexes to row color names.mid_rule: bool
    (a) Boolean representing whether the table should have midrules between rows.
12. mid_rule_color: str
    (a) The color name the table's midrule lines should be.
13. link_target: str
    (a) A string representing an href anchor label the figure should link to.

5.1 Important Methods

The Table class was designed to be fully initialized without the neeed of calling additional methods.

To that end, a Table object can be fully constructed by providing only three pieces of information: 1) a pandas dataframe, 2) a cross reference label, and 3) the type of table to create.

Certain use-cases may necessitate creating an ad-hoc table (such as for aligning document elements in non-standard ways) that would benefit fromt he use of the underlying method calls within the Table class. This type of functionality is still being developed.

A notable external function, however, is make_row_colors_dict():

1. make_row_colors_dict(df, in_values, column=None, color=None)
    (a) df: pd.dataframe
        i. The pandas dataframe to be colored
    (b) column: str
        i. The column name to be used for conditional highlighting
    (c) in_values: list or pd.Series
        i. The list of values to search column for, OR a boolean pandas series.
    (d) color: str
        i. A latex interpretable color string (a colorname or colorname!value)

5.2 Example Usage

Creating a table:

```
# Create an example dataframe:
t = pd.DataFrame({'a': [1,2,3], 'b': [4,5,6]})

# Create a color dict to highlight col 'b' values greater than 5:
color_filter = t['b'] > 5
row_colors = make_row_colors_dict(df=t, in_values=color_filter, color='red!25')

# Create the final table

table = Table(
    data=t,
    row_colors=row_colors,
    table_type='table',
    label='colored_table',
    caption='Result',
    link_target='lot'
)

# Multi-Index Table Example:
arrays = [
    ['Colors', 'Colors', 'Colors', 'Fruit', 'Fruit', 'Fruit', '&Meat', '&Meat'],
    ['red', 'red', 'blue', 'apple', 'apple', 'grape', 'chicken', 'chicken'],
    ['Y', 'N', 'Y', 'Y', 'N', 'Y', 'Y', 'N']
]
```

```
tuples = list(zip(*arrays))

tuples = list(zip(*arrays))
index = pd.MultiIndex.from_tuples(tuples, names=['Category', 'Item', 'Flag'])

data = pd.DataFrame({'A': np.random.randn(8), 'B': np.random.randn(8), 'C': np.random.randn(8)

d = data.T
d.index.name = 'rows'

x = pd.DataFrame(columns=d.columns, index=data.index)
```

Table 1: Result

	a	b
0	1	4
1	2	5
2	3	6

6 Figure

This class represents a latex Figure.

A Figure object an be initialized in one of two ways: 1) A matplotlib.Figure can be passed, or 2) a filepath pointing to a figure may be passed. Figures are placed into non-floating mini-pages by default. This means that figures will always appear in-line in the order they were added to the document.

If a matplotlib.Figure is used to create a LatexPart.Figure object, then a copy of the figure will be automatically exported and saved to a local pdf. This is necessary for the latex compiler to be able to add the image when the document is ultimately rendered.

The Figure class accepts the following arguments:

1. label: str
   (a) A string the figures label - used to cross-reference.
2. figure: obj
   (a) An object representing the figure being added
3. max_height: float
   (a) A float representing the proportion of max text height the figure can be.
4. max_width: float
   (a) A float representing the proportion of max text width the figure can be.
5. graphics_path: str
   (a) A string representing the path to a saved figure (if needed).
   (b) Does not include the filename.
6. caption: str
   (a) String representing the caption to be used with the Figure.
7. empty_label: bool
   (a) A flag used to track whether caption should be empty or not.
8. link_target: str
   (a) A string representing an internal href anchor label.
   (b) Used to link back to table of contents, lists of figures and lists of tables.

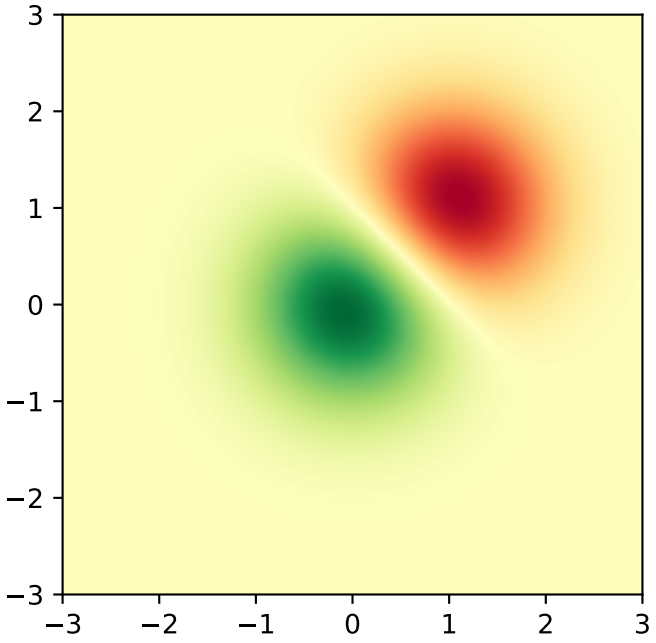
6.1 Example Usage

Creating a figure:

```
# Create Example Figure:
import matplotlib.cm as cm
import matplotlib.pyplot as plt

delta = 0.025
x = y = np.arange(-3.0, 3.0, delta)
X, Y = np.meshgrid(x, y)
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
```

```
Z = (Z1 - Z2) * 2

fig, ax = plt.subplots()
im = ax.imshow(Z, interpolation='bilinear', cmap=cm.RdYlGn,
               origin='lower', extent=[-3, 3, -3, 3],
               vmax=abs(Z).max(), vmin=-abs(Z).max())

# Create the final Figure
figure = Figure(
    figure=fig,
    label='figure_label',
    caption='Here is a caption',
    link_target='lof'
)
```

Figure 1: Here is a caption



7 Section

This class represents a latex Section.

Sections are a common way to organize information within a document.

Currently, the Section class supports Sections, sub-sections, and sub-sub-sections.

The Section class accepts the following arguments:

1. child: LatexPart
   (a) A latexPart to be contained within the section.
2. link_target: str
   (a) A string signifying a cross-reference label to link the section to.
   (b) Default labels include:
      i. toc (table of contents)
      ii. lof (list of figures)
      iii. lot (list of tables)

7.1 Example Usage

```
# Create a Section
section = Section(
    'This is the section name',
    level=1,
    link_target='toc'
)

# Add some content to the section:
section.add(Text('Here is some text that goes into the section.'))
```

8 Environment

This class represents a latex environment.

Environments are used to change the layout of parts of a latex document. For example, centering elements such as text, figures and tables can be done using environments.

Environments are treated as 'containers' - that is that they contain or encapsulate other report objects. To encapsulate a latex element - for example a table - in an environment, simply pass the latex element as the solo argument to an Environment. Once an environment has been declared, additional method calls are used to 'add' the desired enviornments.

Currently, the Environment class supports minipages, landscapes, adjustboxes and centering environments.

The syntax flow is as follows: Create latex element -> add element to environment -> add specific environment behavior.

The Environment class accepts the following arguments:

1. child: LatexPart
   (a) A latexPart to be contained within the Environment.

8.1 Example Usage

```
# Example table:

text = Text(
    'repeat this text' * 5
)

# Declare an environment encapsulating the table from above:
environment = Environment(text)

# Add a centering environment:
environment.add_centering()
```

repeat this textrepeat this textrepeat this textrepeat this textrepeat this text

9 Columns

This class is used to create newspaper-like column layouts. This class serves as a convenient way to ensure document elements appear next to eachother.

The column class accepts a single argument:

1. arg: int / list
   (a) An int specifying the number of columns, or a list containing each columns' contents.

9.1 Example Usage

```
# Example:

# Make  Table
table = Table(
    data=pd.DataFrame({'A':[1,2,3], 'B':[3,4,5]}),
    table_type='tabular',
    label='example'
)

center_table = Environment(table)
center_table.add_centering()

# Create a Column object and fill column contents:
columns = Columns(3)
columns[0] = Figure(figure=fig, label='figure', max_width=1.0)
columns[1] = center_table
columns[2] = Text('some random text ' * 20)

or

columns = Columns([
    Figure(figure=fig, label='figure', max_width=1.0),
    center_table,
    Text('some random text ' * 20)
])
```

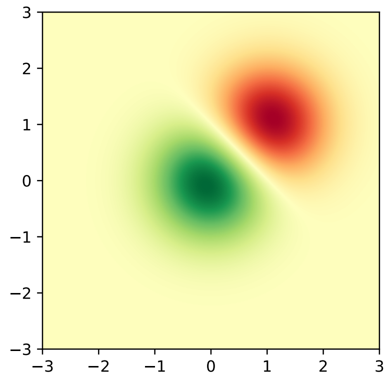


Table 2: None

	A	**B**
0	1	3
1	2	4
2	3	5

some random text some random text