The `tufte-style-article` class

Sylvain Kern

March 1, 2021

https://github.com/sylvain-kern/tufte-style-article

`tufte-style-article` is a LaTeX class with a design similar to Edward Tufte's works. His designs are known for their simplicty, legibleness, extensive use of sidenotes in a wide dedicated margin and tight text and graphic integration. This class is however not a rigourous copy of E.Tufte's works, it is more of an inspiration. It also includes design features from *The Elements of Typographic Style*.[1]

This documentation gives a glimpse of what the class looks like and does, while explaining how to install and use it. I tried to make it as complete as I could; some parts can still be unexplained or so do not hesitate to ask me if something remains unclear. I tested it on several LaTeX distributions, but it can still spit unexpected errors. Feel free to ask me for information or report a malfunction if you encounter one!

I am aware that numerous E.Tufte-based classes exist around the Internet, I just wanted to create my own to really feel and internalize this design grammar. Eventually, this is just my view of what I find well-presented and eye-pleasing in a document and not prescriptive rules and guidelines. Everybody can feel free to adapt, customize, or contribute to this class.

Before I get into the subject, I want to thank the members of tex.stackexchange.com, who are basically an endless source of knowledge. Without this forum I would have gave up on the very first issue I encountered.

Edward Tufte is a statistician, computer scientist and professor at Yale University. His personal website: www.edwardtufte.com

[1] ROBERT BRINGHURST, *The Elements of Typographic Style*, 1999.

See section 4 for known issues, questions about this class and bug reporting.

I give some information for contributors on section 5.

Contents

1 *Installation*

This class' source file is `tufte-style-article.cls`, avaliable on the following repository:
`www.github.com/sylvain-kern/tufte-style-article`

The file can just be put in the same folder as your main `.tex` file. Overleaf users will have to do this, since it does not support custom class installation. For Windows or Linux users with an installed LaTeX distribution, please see respectively the two following sections, on how to install `tufte-style-article` on your system.

In order to make the code environments[2] and syntax highlighting work, it is needed to have Python[3] installed on your system, along with the `pygments` package. With `pip` simply execute

[2] See section 10.

[3] Python has to be on the `PATH`.

```
pip install pygments
```

MiKTeX users on Windows

1. Create a `localtexmf`[4] directory if you do not already have one, for instance

   ```
   C:\Users<you>\localtexmf
   ```

2. Create a `tex\latex\` directory in the `localtexmf` one, and inside it, create a folder named `tufte-style-article`.

3. Paste the `tufte-style-article.cls` file in that `tufte-style-article` folder and you should be good. Eventually, the class file is located at

   ```
   C:\Users\<you>\localtexmf\tex\latex\tufte-style-article\tufte-
   ↪  style-article.cls
   ```

4. Open MiKTeX console, go to `Settings`, `Directories` tab. Click on `add`, and enter yout `texmf` path.

   ```
   C:\Users<you>\localtexmf
   ```

5. Finally, go to the `tasks` tab, and hit `Refresh file name database`.

`tufte-style-article` is now installed on your system ! MiKTeX will recognize and find the class file without it having to be in your project folder.

[4] More on `texmf` and how to install custom classes and packages on MiKTeX here: `https://tex.stackexchange.com/questions/10498/installing-a-class`.

TEX Live users on Linux

1. Create a `localtexmf`[5] directory if you do not already have one, for instance

```
$HOME/.texmf
```

2. Create a `tex/latex/` directory in the `.texmf` one, and inside it, create a folder named `tufte-style-article`.
3. Paste the `tufte-style-article.cls` file in that `tufte-style-article` folder and you should be good. Eventually, the class file is located at:

```
$HOME/.texmf/tex/latex/tufte-style-article/tufte-style-articl
↪  e.cls
```

4. Update the `texmf` with

```
mktexlsr $HOME/.texmf
```

5. Check if it worked with

```
kpsewhich tufte-style-article.cls
```

`tufte-style-article` is now installed on your system ! TEX Live will recognize and find the class file without it having to be in your project folder.

[5] More on `texmf` and how to install custom classes and packages on TEX Live here: https://tex.stackexchange.com/questions/96976/install-custom-cls-using-tex-live-in-local-directory.

2 *Presentation and Usage*

This section has come quite thick, so a cheat sheet should come soon to summarize all this.

Class definition and options

This class is named `tufte-style-article`. The preamble is therfore written as follows.

```
\documentclass[<options>]{tufte-style-article}
```

It is inherited from the `article` class, so all the options of the latter fit in `tufte-style-article`'s options. There are also new ones for this class, which are:

`raggedright`	Makes all paragraphs align on the left without right-justification, as it is the case in this very document.
`parskip`	Separates paragraphs with a vertical space instead of indenting so that all text is rigorously left-aligned.
`noheaders`	Deletes the current section reminder on page header, just displays the page number on the top outer corner.
`casual`	Makes all sections numberless. Puts them natively in the toc anyway.
`sans`	Turns the font to sans serif Source Sans Pro, for extreme casualness.
`colorful`	Like in this document, makes titles, figure labels and note numbers colored. The accent color is defined by `main_accent`.
`notufte`	Remove margins. Turns sidenotes to footnotes and makes figure captions appear under them. Appropried for small casual reports or for `pandoc conversion`.

E.Tufte prefers left over full justification because it reduces the variation of spaces between words. The irregularities on the right makes the lines also easier to follow. However, R.Bringhurst fully justifies the main text in his *Elements*, so I decided to give this choice to the user. Both indent paragraphs on the first line, except just after title headers.

The big margin

It may noticeable that there is a big margin running all through the document, a design feature present in all E.Tufte's works but also in the *Elements of Typographic style*. I find this design –a *1.5-column* setup– to have many advantages over a regular one-column setup. Here are a few reasons why.

- The main text block has a reduced width of about thirteen words per line, which makes the eyes follow the lines easier.
- The layout is less constrained thanks to the negative space freed in the margin.
- The margin can be used to place elements that would break the main prose, such as sidenotes,[6] captions of figures, tables and other stuff, and even small figures. This tidies up the main text area while making margin stuff immediately noticeable, but not disturbing.
- The overall text-image inclusion comes tighter and more natural.

All in all, this design is neither too crowded as everything is at its place, nor too empty due to huge blank margins.

To insert a numbered margin note, use `\marginnote{<your note>}`. This gives the following result in the margin.[7]

To insert an unnumbered piece of text in the margin, use `\margintext{<your text>}`, which gives the following result in the margin.

All pieces of text in the margin are in `\footnotesize` and `\raggedright`, as defined in this class' macros.

[6] Hello there!

[7] This is a numbered note.

This is unnumbered margin text.

To insert raw, unformatted text or graphics or whatever in the margin, use the command `\marginpar{<unformatted margin text>}` and it will look like what appears here in the margin. Note how it is the same size as the main text. Also LaTeX tries to fully justifiy it which is hard on such a small width.

This is just unformatted text in the margin. It is in `\normalsize`, which makes it stick way too much out.

Paragraphs

The main text is structured in paragraphs. They can be left-aligned as it is the case here, or fully justified according to your taste, depending on the given class options. Likewise, the paragraphs are whether indented or separated with a vertical space.

The indents are one em wide, *i.e.* the size of the font in pt. If you choose a 11 pt size in the options declaration of the class, then the indent will be 11 pt wide.

This one and the following are paragraphs with vertical space separation. It works better for documents not intended to be read linearly, or when there is few text compared to figures, equations or anything which does not fit in the prose.

The vertical space looks like this, with a separation of one em. This is just some more text to fill the paragraph, and give a glimpse of the overall look. Quick reminder, load the vertical separation with the `parskip` option in the class definition.

There is a way to make paragraphs stretch all the way to the margins, like this one. See how it continues and reaches for the most outer or inner margin. It also works for two-sided documents so that for odd pages it stretches to the right, and for even pages it stretched to the left. Of course, side notes might be difficult to use here, and I do not predict how `\marginnote` and `\margintext` act here, but it may become handy to have an environment stretch a bit more than the regular text span.

To stretch the main text area to the margins, use the following environment.

```
\begin{wide}
<...your content will be displayed in a wide mode...>
\end{wide}
```

LaTeX may not get the formatting right upon first compilation, so if that occurs, just re-execute the compiling program and it should work properly.

Fonts

The main serif font is Palatino, loaded with the `mathpazo` package. I find it really legible and well-balanced, while being less harsh than Computer Modern, the default LaTeX font. `mathpazo` has full math support too; here are some examples:

Although E.Tufte uses Monotype Bembo as his main serif font, Palatino seems to be a fair alternative, from the same family, and easy to get with LaTeX.

$$e^x = \sum_{n=0}^{+\infty} \frac{x^n}{n!} \; ;$$

$$\frac{\hat{\vec{p}}^2}{2m} |\Psi(t)\rangle + V\left(\hat{\vec{r}}, t\right) |\Psi(t)\rangle = i\hbar \frac{\partial}{\partial t} |\Psi(t)\rangle \; ;$$

$$x_{\pm} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \; ;$$

$$\int_{-\infty}^{+\infty} e^{-\alpha x^2} \, \mathrm{d}x = \sqrt{\frac{\pi}{\alpha}}.$$

$$\psi(\vec{r}) = \frac{j}{\lambda} \iint_S \psi(\vec{r}_0) \frac{e^{jk|\vec{r} - \vec{r}_0|}}{|\vec{r} - \vec{r}_0|} \mathrm{d}S. \tag{1}$$

The sans-serif font is Gillius from the `gillius` package, which is almost identical as Gill Sans, E.Tufte's choice for sans-serif and book titling. It does not clash with Palatino while being elegant and readable. It particularly suits for titles or page headers as it can be seen on this document.

The `mono` font is Droid Sans Mono, from the `droidmono` package. It has a more of a sans-serif style unlike `Courier` or `Computer Modern Teletype`, LaTeX's default mono font. I find it lighter and more adapted for code snippets. The typographic gray is also about the same as Palatino, so that little urls, emails or code references typed with Droid Sans will not stick out in the serif text[8].

For examples of code writing with this class, see section code.

[8] If you want inline pieces of code to stick out, don't worry, macros are provided. See section 10.

Figures, tables and stuff

Images in figures

Edward Tufte's designs are known to be really tight when it comes to including images with text. The main pet peeve I had with one-column designs is when I included a small figure in the document, it had to visually break the text and generate large unpleasing blank spaces. Also, more often than not, the text width is too much for the images, resulting in huge one-liner captions for very small figures.

The 1.5-column design fixes this by putting all captions in the margins, as well as small enough figures, which tidies the document a lot.

Figure 1. 1919 map of the Finistère in French Brittany. This figure is in the main text column, with a caption in the margin aligned with the top of the image. For images with width less than the text width, they will be outer-aligned so that they are close to the caption.

To put a graphics in the text like in the figure 1, use

```
\textfig[<optional width>]{<file path>}{<caption>}{<label>}
```

The `<optional width>` is a number between zero and one wich determines the image width relative to the text width. The default value is 1, like on the figure 1.

The same macros are provided for images in the magins and wide images, respectively shown in figures 2 and 3:

```
\marginfig[<optional width>]{<file path>}{<caption>}{<label>}
\widefig[<optional width>]{<file path>}{<caption>}{<label>}
```

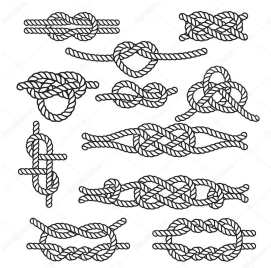
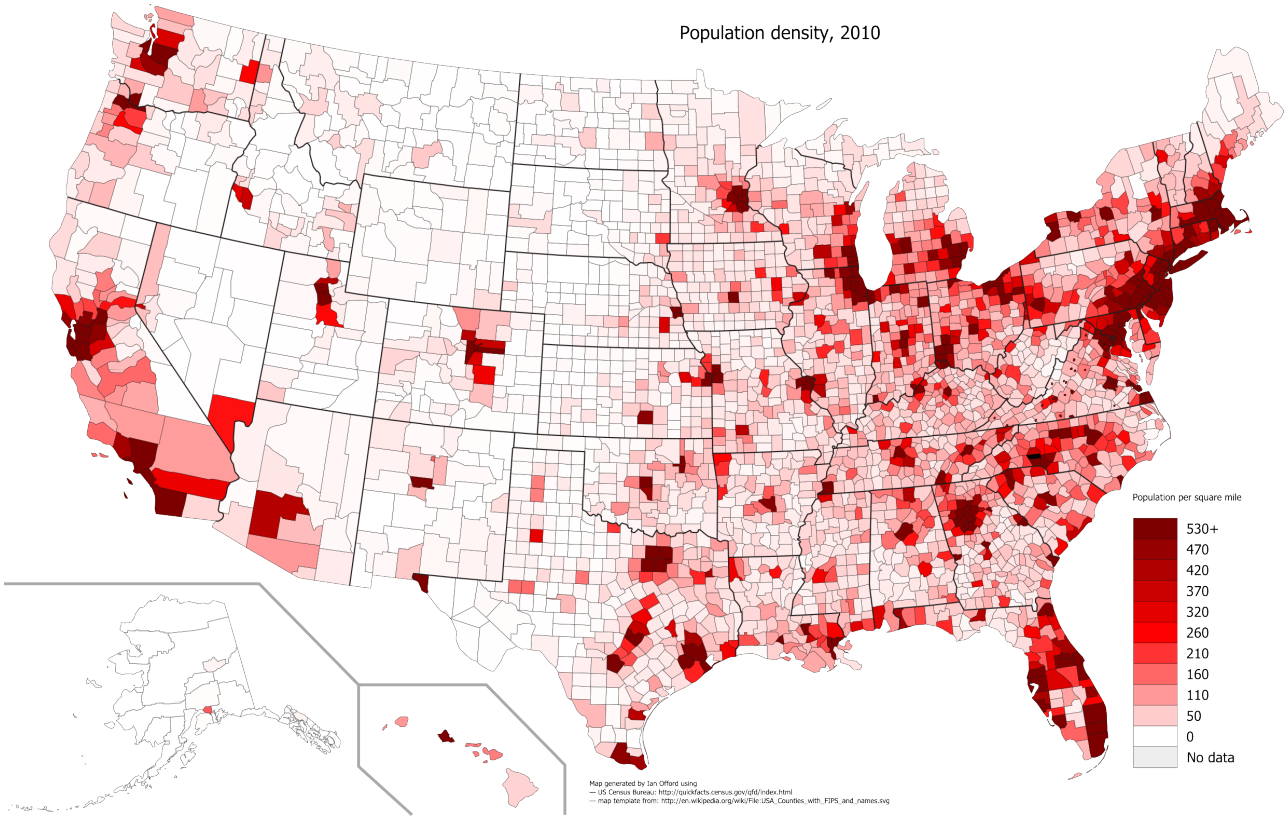


Figure 2. The most common sea boat knots. This image can be displayed rather small, so it fits in the margin. The caption is displayed below.



If for any reason a figure caption has to be put in the main text block, no worries, the following macros will do. The result of `\plainfig` is shown in figure 4.

```
% plain figure with textwidth
\plainfig[<optional width>]{<file path>}{<caption>}{<label>}
% plain figure with full width --like \widefig
\plainwidefig[<optional width>]{<file path>}{<caption>}{<label>}
```

Figure 3. The US census map from data collected in 2010 – `www.ecpmlangues.u-strasbg.fr` This is a wide figure, stretching from the innermost to the outermost margin.

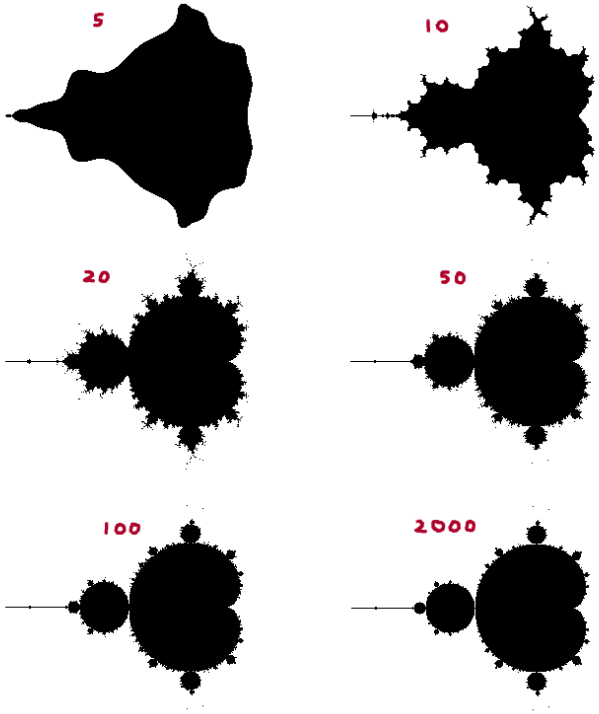


Figure 4. The Mandelbrot set with different depths of iteration. This caption is not in the margin but in the main text area. It can sometimes be useful, for example with really really long captions.

The figures generated by all these macros do *not* float: LaTeX will place them exactly where the command is.[9] The integration will not work 100% of the time, especially if there are too many figures in comparison to the amount of text.

[9] If the command appears to be at the end of a page, the figure may be thrown at the beginning of the next page, with the margin text still on the current page.

The positions of the macros are to be fine-tuned once the document is completely written, in order to have full control of the text-graphic integration. Figures may be moved to prevent overcrowding the margin, or some `\newpage` can be inserted when figures need to be on the next page, for instance when an image is rejected to the next page while its caption is still on the current page.

Tables

As tables can change by a lot, I do not have provided macros for quick table formatting. However, there are tools for a table placement similar to the figures.

Code

Code can be inserted, whether with simple code boxes or captioned snippets that look like the following.

```c
int main(int argc, char *argv[]) {
  printf("Hello world!");
  return 0;
}
```

Listing 1. Hello world in C. This is a captioned code snippet.

The background is a light gray that helps make the code stick out just enough without distracting the eye too much. The code itself is syntax colored according to the used language. There are several environments for code boxes, explained below.

For a simple code box with neither line numbering nor caption, the macro environment is the following.

```
\begin{codebox}{<your language in lowercase>}
<
Your code. It can contain all special characters such as { } ( ) [
↪  ] \ % (the percent mark here is grayed just because on this
↪  very code box the language is set on latex)
It break lines.
It will end only when reading the %\end{codebox} below.
>
\end{codebox}
```

This supports most of the classic languages. Here are some examples for the language option: `c`, `c++`, `python`, `java`, `latex`...

If a specific language is not recognized, use the `text` option instead: it will display the code raw, without syntax coloring.

For a code box *with* line numbering –still without caption– use the following environment.

```
\begin{codeboxnum}{<your language in lowercase>}
<your code>
\end{codeboxnum}
```

For captioned code snippets, the same environments exist, as shown as follows. For example, the listings 1 and 2 are respectively unnumbered and numbered code snippets.

```
\begin{snippet}{<language>}{<caption>}{<label>}
This code will be displayed in a captioned code box, without line
↪  numbering.
\end{snippet}

\begin{snippetnum}{<language>}{<caption>}{<label>}
```

```
This code will be displayed in a captioned code box, with line
↪ numbering.
\end{snippetnum}
```

Small pieces of code can be useful to put inline in the main text flow. This class provides a command to things like this: `public int size() {}`. Use the following to insert a piece of code in the text.

If the piece of code inside the `\inlinecode` contains curly braces, use another character to delimit your code, the same at beginning and end. Underscore (_) and pipe (|) will do fine.

```
\inlinecode{<language>}{<your code>}
% if there are curly braces in your code
\inlinecode{<language>}_<your code>_ % or
\inlinecode{<language>}|<your code>|
```

`\inlinecode` does not break at lines, so be careful, it can sometimes protrude on the right margin. If it is the case, go to a new line by inserting `\\` just before `\inlinecode`.

The following chunk is an example snippet to show the look when the code is a bit heftier. See how the box breaks at the end of the page.

Listing 2. A source code snippet of 29jm's amazing SnowflakeOS! This is a numbered code snippet that goes through several pages.

```
1  #include <kernel/multiboot2.h>
2  #include <kernel/sys.h>
3
4  static const char* tag_table[] = {
5    "TAG_END",
6    "TAG_CMDLINE",
7    "<unknown>",
8    "TAG_MODULE",
9    "TAG_MEM",
10   "TAG_BOOTDEV",
11   "TAG_MEMMAP",
12   "TAG_VBE",
13   "TAG_FB",
14   "<unknown>",
15   "TAG_APM",
16   "<unknown>",
17   "<unknown>",
18   "<unknown>",
19   "TAG_RSDP1",
20   "TAG_RSDP2",
21 };
22
23 /* Prints the multiboot2 tags given by the bootloader.
24 */
25 void mb2_print_tags(mb2_t* boot) {
26   if (boot->total_size <= sizeof(mb2_t)) {
27     printke("no tags given");
28     return;
29   }
30
```

```
31    mb2_tag_t* tag = boot->tags;
32    mb2_tag_t* prev_tag = tag;
33
34    do {
35      const char* tag_name;
36
37      if (tag->type < sizeof(tag_table) / sizeof(tag_table[0])) {
38        tag_name = tag_table[tag->type];
39      } else {
40        tag_name = "<unknown>";
41      }
42
43      printk("%12s (%2d): %d bytes", tag_name, tag->type,
        ↪  tag->size);
44
45      prev_tag = tag;
46      tag = (mb2_tag_t*) ((uintptr_t) tag + align_to(tag->size,
        ↪  8));
47    } while (prev_tag->type != MB2_TAG_END);
48  }
49
50  /* Returns the first multiboot2 tag of the requested type.
51  */
52  mb2_tag_t* mb2_find_tag(mb2_t* boot, uint32_t tag_type) {
53    mb2_tag_t* tag = boot->tags;
54    mb2_tag_t* prev_tag = tag;
55
56    do {
57      if (tag->type == tag_type) {
58        return tag;
59      }
60
61      prev_tag = tag;
62      tag = (mb2_tag_t*) ((uintptr_t) tag + align_to(tag->size,
        ↪  8));
63    } while (prev_tag->type != MB2_TAG_END);
64
65    return NULL;
66  }
```

Advice for text formatting

Compilation

This class compiles with `pdflatex`. I tested it with MiKTEX on Windows, TEX-live on Linux and Overleaf. If you use code boxes or snippets, make sure you compile with the `-shell-escape` flag. Eventually, the following compilation line should work everywhere.

```
pdflatex --shell-escape yourdocument.tex
```

The compilation times can be quite long, especially if there is a lot of heavy code, hopefully it is not a problem for most cases.

I am still working to optimize the class by reducing the compilation time.

3 *Customization possibilities*

4 *Known issues*

In this section I gather the issues that have popped and been reported. I will try to fix them as best as I can. If you spot a malfunction of any kind in this class or you just have a question about all this, feel free to send me an email at:
sylvain.kern98@gmail.com.

5 *Contribute*

6 *Package definition*

7 *Implementation*