

All about the markdowns baby

Keith Chamberlain

18 February 2018

Abstract

This post discusses the installation and first use of Visual Studio Code (VSCode), MiKTeX, Pandoc, Git, Github Desktop and Mendeley. The purpose of installing the software is to enable writing reproducible documents, using Markdown and the statistical programming language of the reader's choice, using VSCode and supporting extensions. These documents can be converted to Word format (docx), PDF, and HTML (to name a few), using one of thousands of predefined citation and document styles available online, with Pandoc (the Swiss Army knife of document converters) and the LaTeX distribution of one's choice, such as MiKTeX. Git is also installed to enable VSCode's functionality with version control, since VSCode is built to work with Git. Mendeley is used as the a bibliographic library synchronization manager between one's local BiBTeX library, and one's bibliographic library on the cloud. As a group, the softwares provide a cross-platform full service documentation and coding kit for languages that RStudio does not yet support, exceeding what can be done with a simple word processor and copy-paste from analytical packages. At the end, scaled PPM difference calculators written in VBA are presented. |

Contents

Installing VSCode, MiKTeX, Pandoc, and Git	3
Visual Studio Code and Markdown	3
Software Installation	4
Extensions for VSCode	7
System PATHs	7
Environment Variables	8
Pandoc version (Windows)	10
Pandoc version (MacOS)	11
MacOS Path	12
Rearrange the MacOS Path	13
Minimal Example To Verify things Work	15
Continuing with the Edu	19
New Projects in VSCode	19
The File Extensions	19
Working with Pandoc-Markdown	19
Markdown Headings	20
Heading 1	20
Heading 2	20
Working with Markdown	20
Support for HTML Tags	20
Heading 2 with Color and an ID	21
Links	21
Inline code	21
Block Code	22
Lists	23
Images	23
Citations	23
Mendeley Prereq	24
Metadata	24
nocite	25
Markdown Summary	25
The VBA code That Was Promised	26
diff()	26
classicPercentDiff()	28
massDiff()	28
refDiff()	29
References	30

Installing VSCode, MiKTeX, Pandoc, and Git

www.ChamberlainStatistics.com

This author was not planning to do much with Microsoft's Visual Basic for Applications (VBA; 2016) other than to introduce Excel's VBA editor. The purpose of introducing the editor was to show only how to generate simulated random data similar to the data generated in R and Python, on the blog of [ChamberlainStatistics](#), and then to integrate Python into Excel using the Python package [xlwings](#) (Zoomer Analytics LLC, 2018). Documenting the VBA code requires something akin to [R Markdown](#) (RStudio Team, 2015, 2016a) that this author used to notebook R and Python code in earlier posts. The limitations for VBA in R Markdown was that of no highlighting of syntax, and no notebooking. As a result of those limitations, this author was not planning on continuing to use VBA, since it did not easily assimilate into this author's workflow.

Not only did this author continue to use VBA to write some User Defined Functions (UDFs), I also installed another code editor besides [RStudio](#) (RStudio Team, 2016b), in order to use **VBA code blocks** written in [Markdown](#) with syntax highlighting. VBA highlighting is not *yet* supported by [R Markdown](#), RStudio's specific flavor of Markdown.

Readers may have noticed that on previous posts that contained VBA code, comments were marked twice. The first mark was a pound sign: #, the single line comment token in R and Python that is recognized by R Markdown. The second mark was the actual VBA comment marker, ', the apostrophe. Without the pound signs, strings and comments indicated using VBA syntax 'broke' in R Markdown. The pound sign was the workaround that enabled, at the very least, comment highlighting in R Markdown code blocks without resorting to writing support for VBA syntax highlighting. When copying code to the VBA editor from those posts, users still had to delete the pound signs for the code to compile. Enter [Visual Studio Code](#) and a series of programs that work with it.

Visual Studio Code and Markdown

If users are trying to write Markdown with syntax highlighted code blocks, chances are a powerful and light-weight code editor is desired. There are many editors to choose from. After some searching, this author installed a few programs together: Microsoft's [VSCode](#) (Microsoft, 2018), the editor and viewer; [Git](#) (Chacon & Straub, 2014), for markdown and software version control; [Github Desktop](#) (Github Inc., 2018) for a convenient graphical user interface (GUI) to Git; the [MiKTeX LaTeX distribution](#) (Schenk, 2018), for preparing documents, such as deep support of PDFs and their formatting; and [Pandoc](#) (MacFarlane, 2018), a Swiss-Army knife document converter to convert between, e.g., Markdown and HTML, PDF, and docx while applying styles and rendering citations.

The reasons are fairly straight forward for selecting the above software, and specifically VSCode. Something was needed that could show Markdown with embeded VBA or Visual Basic code blocks, LaTeX math, and code from other languages (e.g. C, R, Python), that

resulted in output that could be styled, and that functioned with bibliographies and heading structures, at the right price point. In addition, document conversion from Markdown to PDF, HTML and Docx, and version control was preferred. *VSCode* appears to handle the software development end. **Git** handled the **distributed** version control and interfaced directly with **VSCode**. **GitHub Desktop** made for an easier interface to repositories and has just recently been released. **MiKTeX** and **Pandoc** either handle the rest of the requirements natively, and/or have a user base who provide extensions that support the rest of the functionality.

Software Installation

This Blog discusses the installation of the above software, and the setup of a new Markdown project in VSCode that has all of the desired bells and whistles. Software installation is handled first, followed by checking for functionality of Pandoc and adjusting the system PATH, if needed, to find Pandoc, Git and LaTeX. Then a fully functional example, that has minimal explanation to it's parts, will be presented that should verify that the machinery works as intended. Finally, the modest VBA code will be displayed that this author was talking about having taken on.

1. **Common:** [Download and install VSCode](#) or another suitable editor. The link here assumes one is installing on Windows. Other operating systems are supported. For example, the [Mac download here](#)
 - Add some extensions from the [marketplace](#). This author installed the following:
 1. [VSCode VBA](#)
 2. [VSCode VBA icons](#), and,
 3. [VSCode VBA snippets](#); all by Scott Spence.
2. **Windows:** [Download and install Git](#). Again, the link here assumes the install will be on Windows, though other operating systems are supported. This author actually installed Git through VSCode, which asked for Git when opening the program for the first time. It makes sense to use Git instead of having to update the style sheets on one's computer in all of their respective folders manually every time someone out there on the cloud updates their repository.
3. **Common:** [Download and install GitHub Desktop](#) for a graphical front end that is easier than the one provided in Git for setting up and managing repositories. The website should auto-detect one's operating system.
4. **Common:** [Download and install Mendeley Destop](#) for a graphical bibliographic manager that is easy to use, and synchronizes on all of one's devices with the cloud. Note that [Zotero](#) would work fine for this as well.
5. **Common:** [Download and install Pandoc](#), a file converter and interpreter that converts Markdown citations to citations in HTML, PDF and docx, for example.
6. **Windows:** [Download and install MiKTeX for PDF output from Pandoc](#). This

can be skipped if users are going to use Word to print/save other output formats from Pandoc to PDF instead. However, if users are planning on utilizing Pandocs machinery for displaying formulas in Pandoc-Markdown, MikTeX will be needed.

- To avoid confusion, and when prompted during setup, tell setup to let the software install packages on the fly.
1. Once the software is installed, it must be updated. Use the update wizard.
 - The package updater should automatically select the packages that can be updated. If users cannot “Select All” updatable packages during an update, it just means there are other packages that need to be updated first. Just update (“Next”) the ones that are selected already.
 - Run update over again until there are no packages to update.
 - If users still run into trouble, try the Administrator version of the updater tool.
 2. Install the following libraries for PDF citation rendering.
 1. **natbib**. In the **MiKTeX Package Manager** (Start > MiKTeX 2.9 > MiKTeX Package Manager on Windows), or **MiKTeX Package Manager (Admin)** just search for 'natbib' in the Name field to find the package. Highlight the line with the package information, then right-click and select **Install**.

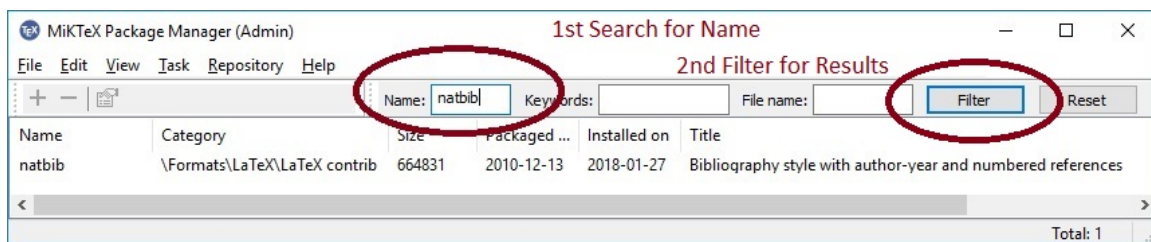


Figure 1: The MiKTeX Package Manager for ‘natbib’ installation.

2. **bibtex**. Search for Name = 'bibtex', and find the package(s) in the list. This author included the 64 bit library as well, in order to match the 64 bit installation of MiKTeX, though I am uncertain if it is necessary. Readers feel free to comment. Install the package(s) as above.
3. **biblatex**. Search for Name = 'biblatex', and find the package(s) in the list. It may help to sort the list by Name first. Install as before.
4. **biber**. Search for Name = 'biber', and find the package(s) in the list. Install as before.
3. And then the following...
 1. **amsfonts** for an extended set of math fonts used for typesetting AMS-LaTeX
 2. **amsmath** the principle package in AMS-LaTeX
 3. **lm** for the “Latin Modern” family of fonts that are public domain from AMS.
 4. **unicode-math** for the complete implementation of unicode maths for XeLaTeX and LuaLaTeX.
 5. **ifxetex** for a logical test of whether one is running under XeTeX.

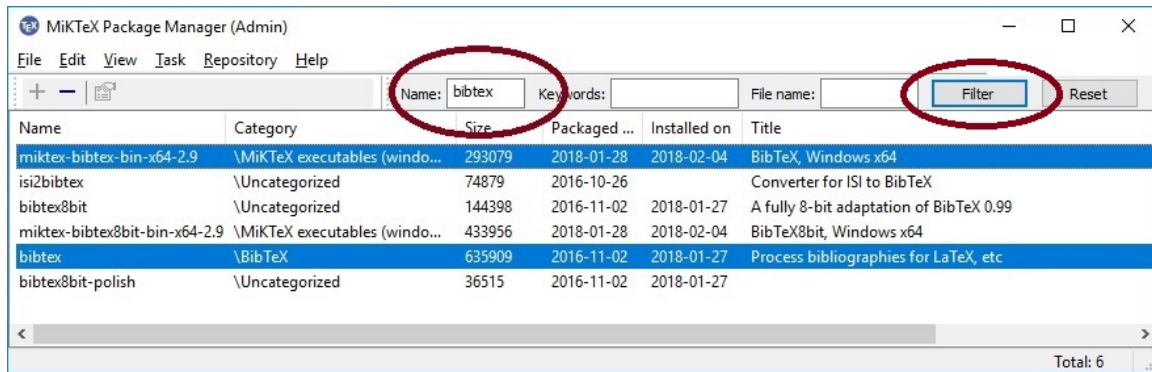


Figure 2: The MiKTeX Package Manager for ‘bibtex’ installation.

6. **listings** for typesetting source code.
 7. **fancyvbt** for verbatim text support.
 8. **longtable** to allow tables to break across pages.
 9. **The oberdiek bundle**
 10. **booktabs** to enhance the quality tables in LaTeX and is compatible with **longtable** (above).
 11. **hyperref** for supporting hypertext.
 12. **xcolor** expands driver specific color capabilities.
 13. **ulem** for package underlining.
 14. **geometry** to customize page layout.
 15. **setspace** to set the space between lines.
 16. **babel** for multi-language support.
 17. **fontspec** for advanced font selection.
 18. **polyglossia** for an alternative to ‘babel’.
 19. **xecjk** for support for CJK documents in LaTeX.
 20. **bidi** for bidirectional typesetting using XeTeX.
 21. **mathspec** for specifying arbitrary math fonts in XeTeX.
 22. **upquote** for setting apostrophes according to “Computer Modern Typewriter” (bent and straight, ‘ and ’).
 23. **microtype** for font micro-typographic extensions introduced by PDFTeX.
 24. **csquotes** for context sensitive quotes.
7. **MacOS: Download and install MacTeX** The MacTeX distribution contains everything needed. Alternatively, one can install the **BasicTeX distribution**, and install the other packages as needed by following the directions on the **Pandoc website under macOS**.
1. Once the distribution is installed, it must be updated, just as with MiKTeX on Windows.
 2. Launch the **TeXLive** utility. This should be in the **Launchpad**.
 1. Once **TeXLive** is launched, click on the icon in the doc and select **Keep in dock** so that the program can be utilized in the future.
 3. The utility will take some time to figure out what packages have updates. The program will come up with a list that, for each package, says “remove”, “Update

- available”, or “Not installed”.
4. **Sort** the list by **Status**, and highlight all packages that have a status of “Remove” or “Update available”.
 5. **Option-click** and select **Update selected**.
 1. The “Remove” packages that are selected are no longer being maintained and will be uninstalled.
 2. The “Update available” packages will be updated.
 3. The “Not installed” packages will be left alone.
 6. Run TexLive until there are no more package updates to install.

Extensions for VSCode

Just as how extensions were installed for VSCode from the [marketplace](#), **Extensions for VSCode** can also be installed from **within** VSCode itself. Click **View** then **Extensions**, or (on Windows) while holding down the **ctrl** and **shift** keys at the same time, press **x** (**ctrl+shift+x**). The following extensions were installed from VSCode. Note that they could have also been downloaded and installed from the [marketplace](#) as well:

1. The [Markdown Extension Pack](#) which installs several markdown extensions to use Markdown in VSCode.
2. The [Markdown Theme Kit](#) which installs VSCode themes.
3. [Markdown + Math](#) for using VSCode for typesetting and rendering TeX math.
4. [LaTeX Workshop](#) to install LaTeX tools for VSCode.
5. [Word Count](#) to show the number of words that in the editor, down in the status bar at the bottom of the VSCode window.
6. [Python Extension Pack](#) that extends VSCode to handle Python.
7. [R](#) to support for the R language in VSCode.
8. [R Development](#) to add support for R code snippets in VSCode.
9. [Pandoc Format](#)
10. [vscode-pandoc](#) to be able to render to additional formats in VSCode, including Microsoft’s docx format.

In VSCode’s Extensions search, just single click on the extension and a browser-tab for the marketplace page will open. Then, if the description sounds useful, click install, and tell VSCode it’s OK to continue. When finished, users may need to restart VSCode.

System PATHs

In order to make it so that VSCode can find Pandoc, which in turn can find LaTeX, etc., it is important that users can verify that some environment variables have been set, which tells the operating system where to look for programs that are not in the current folder; aka that the programs are in the system’s “PATH”. Windows, Unix, Linux and MacOS all have a PATH.

Environment Variables

The following is **Windows specific**. The process of setting the PATHs on Windows 10 is discussed.

1. On a Windows system, open an Explorer window like searching for a file.
2. **Right-Click** on **This PC** (Windows 10) or **My Computer** (Earlier versions) and then select **Properties**.

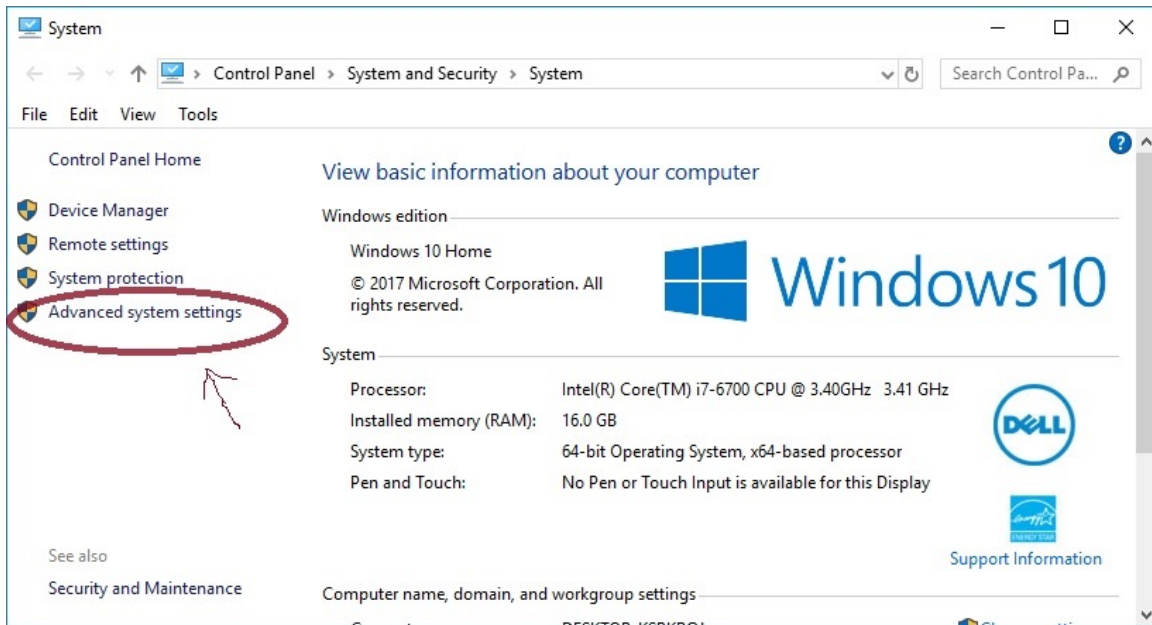


Figure 3: Opening Advanced System Properties from System Information

3. In the **System Properties** window, select **"Advanced System Settings"**. Another window will open.
4. Select the **"Advanced"** tab if it is not selected already, then select **"Environment Variables"**. Another window will open:
5. Select **"PATH"** then select **"Edit"**, and a final window will open.
6. Select **"New"** to add lines to the system PATH and enter the folder name and folders leading to it. Note that in the above PATH-lines, one must supply full-paths (includes the drive letter, and every sub-folder), as opposed to "relative" PATHs. If not already added, add:
 1. C:\Program Files\Microsoft VS Code\bin
 2. C:\Program Files\MiKTeX 2.9\
 3. C:\Program Files (x86)\Pandoc\
 4. C:\Program Files\Git\
 5. C:\Users\YourUsername\AppData\Local\GitHubDesktop\bin

Where "yourUserName" is replaced by the readers user name.

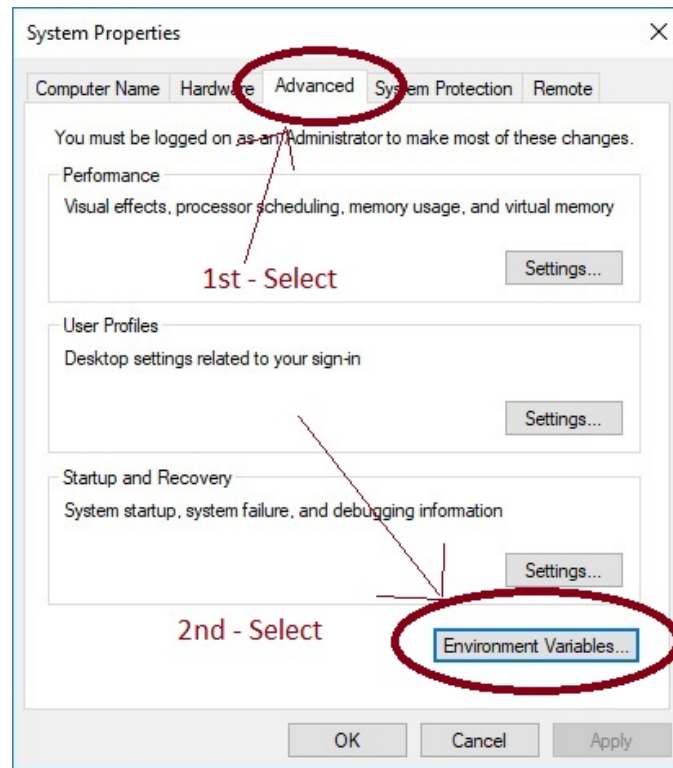


Figure 4: Opening the Advanced System Settings from System Properties

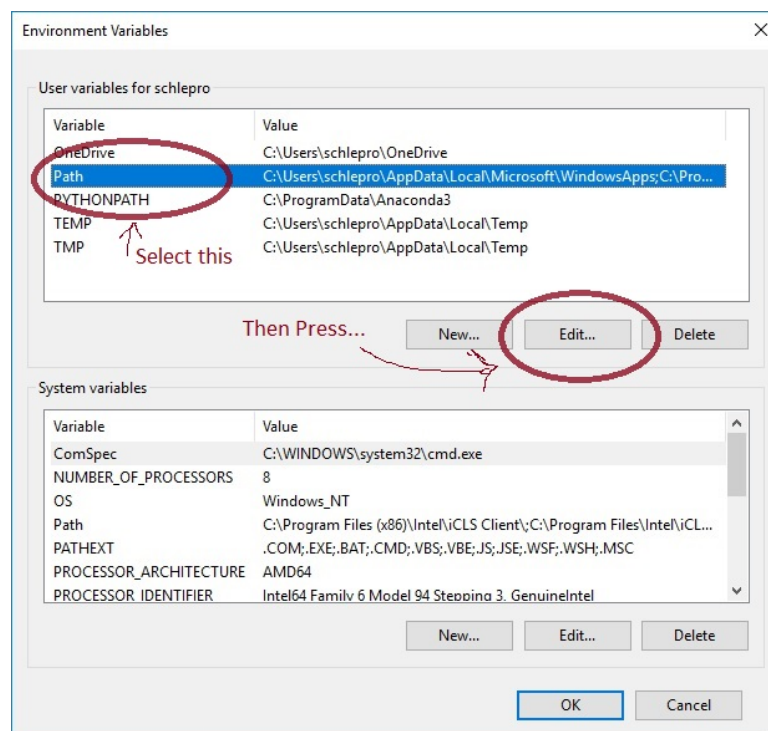


Figure 5: Selecting the PATH variable from the Environment Variables

Pandoc version (Windows)

The following is **Windows specific**. In order to check that the Windows system can find Pandoc, open a command shell.

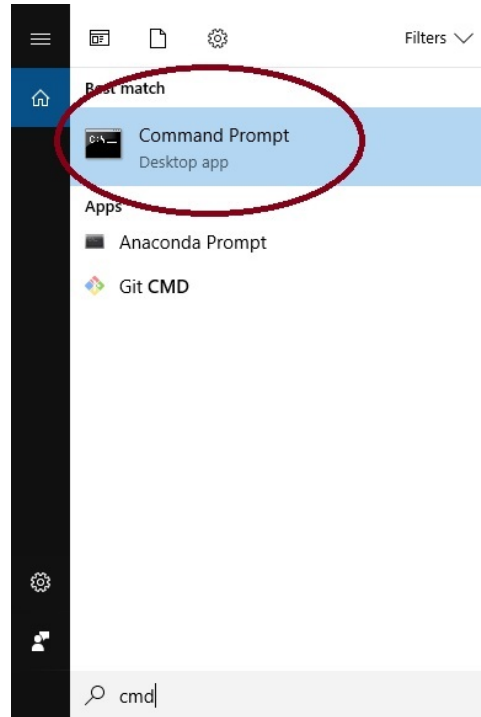
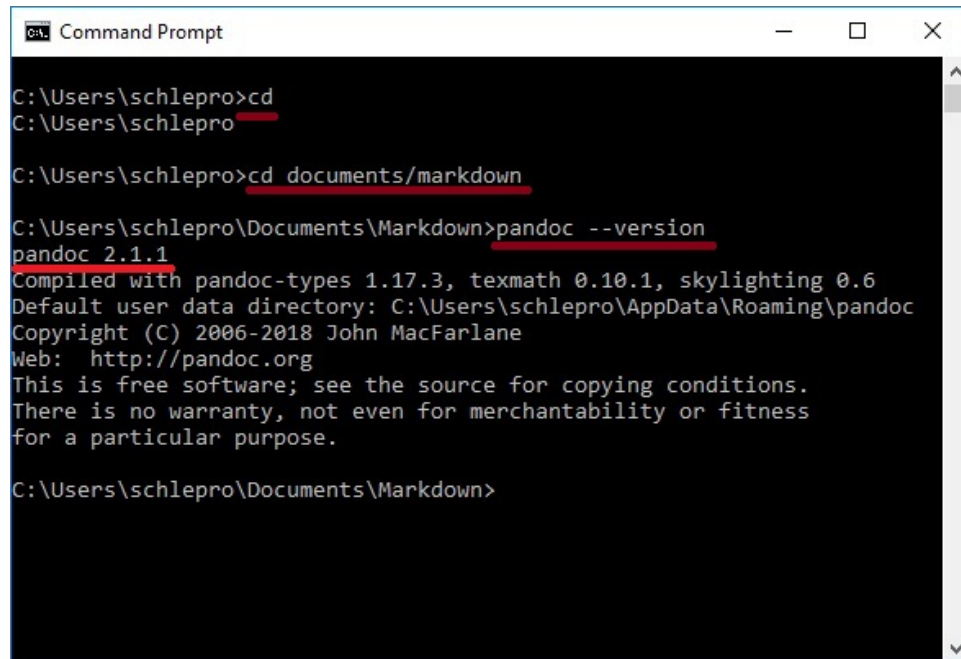


Figure 6: Opening the command shell on Windows

1. Search > type "cmd"
2. Select Command Prompt and a new window will open.
3. The prompt will read like the following:
C:\Users\YourUsername>
4. At the prompt, type `pandoc --version`. Readers should see something like the following:

```
168 > pandoc --version
169 pandoc 2.1.1
170 Compiled with pandoc-types 1.17.3, texmath 0.10.1,
171 skylighting 0.6
172 Default user data directory: ...
173 Copyright (C) 2006-2018 John MacFarlane
174 ...
```



```
C:\Users\schlepro>cd
C:\Users\schlepro

C:\Users\schlepro>cd documents/markdown

C:\Users\schlepro\Documents\Markdown>pandoc --version
pandoc 2.1.1
Compiled with pandoc-types 1.17.3, texmath 0.10.1, skylighting 0.6
Default user data directory: C:\Users\schlepro\AppData\Roaming\pandoc
Copyright (C) 2006-2018 John MacFarlane
Web: http://pandoc.org
This is free software; see the source for copying conditions.
There is no warranty, not even for merchantability or fitness
for a particular purpose.

C:\Users\schlepro\Documents\Markdown>
```

Figure 7: Verifying communication with Pandoc in the command shell

Pandoc version (MacOS)

The following is **Mac specific**. If one has Anaconda3 installed, then Anaconda also has its own version of Pandoc. This version interferes with the newly installed version of Pandoc because the system keeps calling from the old location first. The problem is that the PATH has the Anaconda3 folder listed in the path before `/usr/local/bin`, where Pandoc 2.1.1, the latest version of Pandoc as of this writing, is located. To workaroud, there are two options. (1). **Rename the Anaconda3 Pandoc files** (Pandoc and Pandoc-citeproc) to have `.old` extensions., and/or (2). **Rearrange the system path** so that the Anaconda part of the path comes last. Rearranging the system path is the least intrusive of the two. This author does not yet know the side effects of these changes to Anaconda's functionality.

First, one needs to see if Pandoc can be seen, and what version it is.

1. Click LaunchPad > Other > Terminal, then a **bash** terminal window opens
1. At the prompt type `cd` and then press **Enter** to get to the home directory.
2. At the prompt type `pandoc --version` and press **Enter**.
3. If one reads the following at the prompt in the bash terminal shell, then the workaround that follows it applies.

```
188 $ cd
189 $ pandoc --version
190 pandoc 1.19.2.1
191 Compiled with pandoc-types 1.17.0.4, texmath 0.9,
192 skylighting 0.1.1.4
```

```

keithchamberlain$ cd
keithchamberlain$ echo $PATH
/Users/keithchamberlain/anaconda/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin
keithchamberlain$ pandoc --version
pandoc 1.19.2.1
Compiled with pandoc-types 1.17.0.4, texmath 0.9, skylighting 0.1.1.4
Default user data directory: /Users/keithchamberlain/.pandoc
Copyright (C) 2006-2016 John MacFarlane
Web: http://pandoc.org
This is free software; see the source for copying conditions.
There is no warranty, not even for merchantability or fitness
for a particular purpose.
keithchamberlain$ nano .bash_profile

```

Figure 8: The bash Terminal showing the PATH out of order, and the wrong version of Pandoc

```

193 Default user data directory: ...
194 Copyright (C) 2006-2018 John MacFarlane
195 ...

```

If the Pandoc version reads **something before 2.1.1**, then chances are the latest version of Pandoc is not installed. Even if the latest build of Pandoc is installed, the system may be pointing to the wrong location first. This is the case this author encountered when Anaconda3 is installed on the Mac. If an error is displayed when one attempts to run `pandoc --version` instead, then it could be that the system path has become corrupted. The following instructions will help remedy the problem of the system path as well.

MacOS Path

Verify that the path has all of the elements needed to operate all of the newly installed software.

1. Click on the LaunchPad > Other > Terminal. Once this is done, the **bash** shell will open.
2. In the bash shell, and in order to ensure the current folder is the home folder, type `cd` and press **enter**.
3. In order to check what is already in the path, at the prompt, type `echo $PATH` and then press **enter**. Note that the command is case sensitive, so `echo $Path` will not be looking at the system `$PATH`. The path should look like the following (including the Anaconda 3 part if one has Anaconda 3 installed) where each path entry is separated by a colon, or `:...`

```

208 $ cd
209 $ echo $PATH

```

```

210 /Users/YourUserName/anaconda/bin:/usr/local/bin:/usr/bin:/bin:
211 /usr/sbin:/sbin:/Library/TeX/texbin

```

4. Listed out, the folders are as follows:

1. /Users/YourUserName/anaconda/bin
2. /usr/local/bin # this is where the latest pandoc binaries are located
3. /usr/bin
4. /bin
5. /usr/sbin
6. /sbin
7. /Library/TeX/texbin # this is where the LaTeX binaries are located

Rearrange the MacOS Path

In order to rearrange the system path and/or add the missing lines above, use the `nano` editor.

1. In the home directory, (e.g., in the terminal, type `cd` and press enter) type `nano .bash_profile` and press Enter.

```

230 $ cd
231 $ nano .bash_profile

```

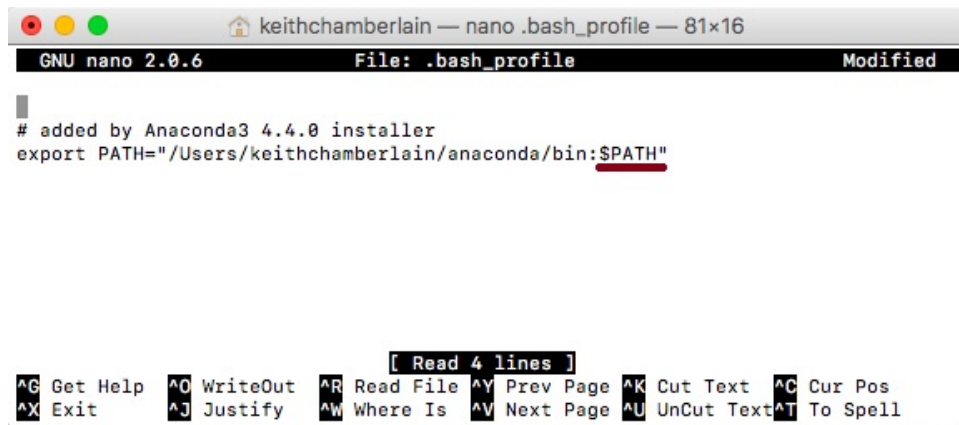


Figure 9: The nano editor before the Anaconda fix

2. A new window will open. This author had the following information already in the `.bash_profile`.

```

# added by anaconda3 4.4.0 installer
export PATH="/Users/myUsername/anaconda/bin:$PATH"

```

3. Edit the line to read such that `$PATH` and any missing pieces to the path come *before* the Anaconda3 addition to the path, as in the following (remember, multiple parts of the path are separated by a colon, or `:`):

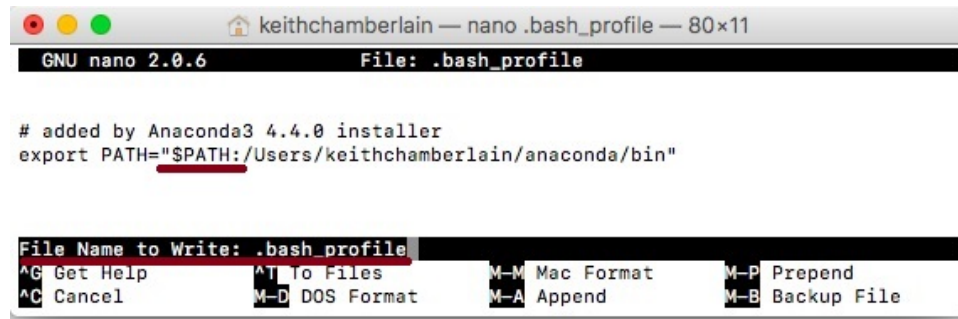


Figure 10: The nano editor and the .bash_profile after being edited

```
# added by anaconda3 4.4.0 installer
export PATH="$PATH:/Users/myUsername/anaconda/bin"
```

4. If the LaTeX binary folder was also missing from the path, then the entry would look like this:

```
# added by anaconda3 4.4.0 installer
export PATH="$PATH:/Library/Tex/texbin:/Users/myUsername/anaconda/bin"
```

5. After making the changes, press **Control+O** and ensuring the file name is correct at the bottom of the window. If it reads exactly `.bash_profile`, then press **Enter** to save. There should be a message about how many lines were written to the file.
6. Then press **Control+X** to exit nano.
7. Exit the terminal altogether by clicking **Terminal > Quit Terminal**. The changes will not take effect until the next time the terminal is opened.
8. Click on the Terminal once again (**Launchpad > Other > Terminal**), then type `echo $PATH` to see the path. One should see (if Anaconda3 is installed) the Anaconda part of the path at the end this time, as follows:

```
255 $ echo $PATH
256 /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin:
257 /Users/YourUserName/anaconda/bin
```

Or **without Anaconda**, the following:

```
265 $ echo $PATH
266 /usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin
```

9. Type the following command at the shell prompt to verify the correct version of Pandoc is visible.

```
270 $ pandoc --version
271 pandoc 2.1.1
272 Compiled with pandoc-types 1.17.3, texmath 0.10.1,
273 skylighting 0.6
274 Default user data directory: ...
275 Copyright (C) 2006-2018 John MacFarlane
```


276 . . .

If `pandoc --version` is 2.1.1 or later, communication with Pandoc in the right folder has been established. Not only that, by ensuring `Library/TeX/texbin` is in the path, readers have also ensured that LaTeX can be found as well.

```

keithchamberlain — -bash — 80x16
Last login: Sun Feb 18 09:41:58 on ttys001
[Keiths-Air:~ keithchamberlain$ cd]
[Keiths-Air:~ keithchamberlain$ echo $PATH]
/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin:/Library/TeX/texbin:/Users/keithchamberlain/anaconda/bin
[Keiths-Air:~ keithchamberlain$ pandoc --version]
pandoc 2.1.1
Compiled with pandoc-types 1.17.3, texmath 0.10.1, skylighting 0.6
Default user data directory: /Users/keithchamberlain/.pandoc
Copyright (C) 2006-2018 John MacFarlane
Web: http://pandoc.org
This is free software; see the source for copying conditions.
There is no warranty, not even for merchantability or fitness
for a particular purpose.
Keiths-Air:~ keithchamberlain$

```

Figure 11: The bash Terminal after the fix to the PATH. ‘`/usr/local/bin`’ is now in the front and the Pandoc version reflects the newly installed version.

Minimal Example To Verify things Work

Now is the time to verify that the nuts and bolts work before getting into specifics about Markdown or file conversion using Pandoc. To get this to work, a bibliographic entry is needed in Mendeley, and Mendeley needs to be set to maintain ones local BibTeX bibliography. Then a [Cascading Style Sheet](#) (CSS; Cascading Style Sheets Working Group, 2018), which describes the style of an HTML document, needs to be downloaded. We will be using one to render a *GitHub like* HTML style. Readers can hit a [tutorial on CSS](#) (W3schools.com, 2018) if interested in learning more. The [Citation Style Language](#) (CSL; Zelle, Bennett, & D’Arcus, 2015) that describes the citation-style for documents also needs to be downloaded. Once downloaded, references to the CSL, CSS and bibliography can be added to the Markdown document’s header. Then conversion to docx, PDF, and HTML is verified.

1. Open Mendeley
 1. Add a bibliographic entry by following the link to [this article on the electron’s mass](#).
 1. On the page, click on the link “**Export Citation**”.
 2. Save the citation to a convenient location on one’s drive. It doesn’t matter if one selects RIS format or BibTeX, just as long as the same format is picked when the reference is imported to Mendeley.
 3. In Mendeley, select **File > Import > bibtex or RIS** and select the file just downloaded.

2. Make sure the Citation Key is noted. For our purposes this author will be assuming the key is Beier2002 (Beier et al., 2002). Note that any unique key can be supplied.

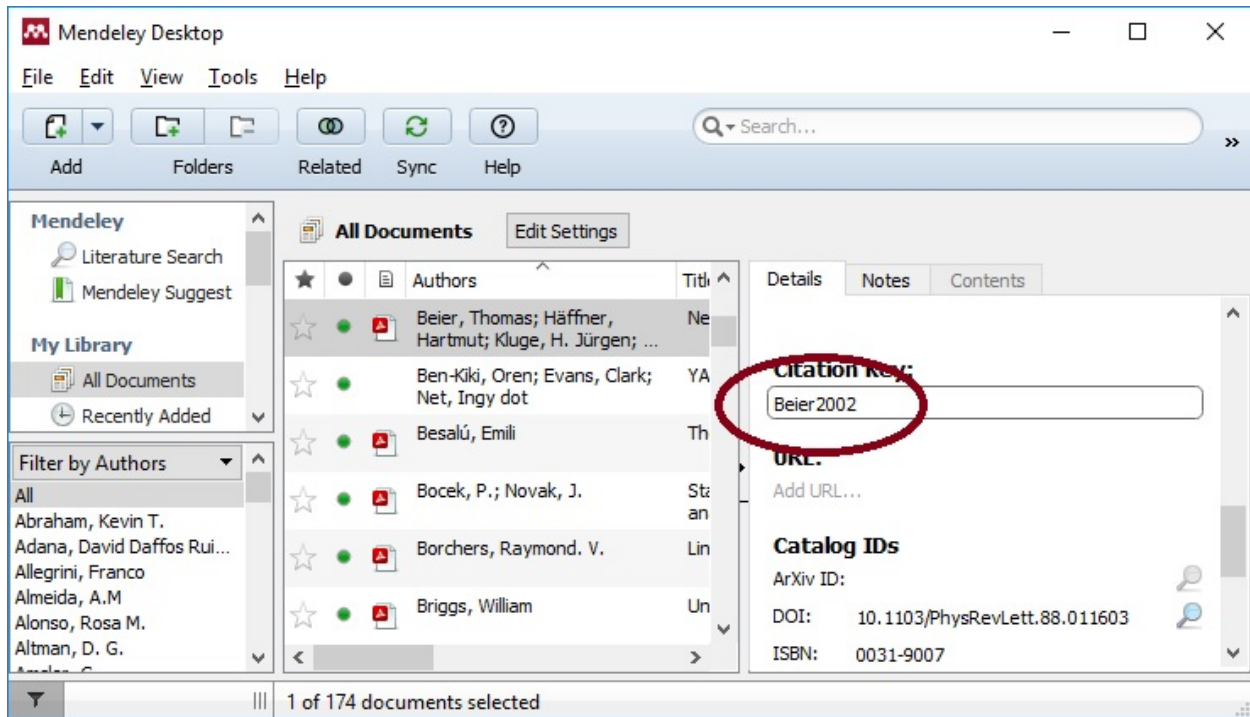


Figure 12: Editing the citation key in Mendeley

3. Still in Mendeley, under Tools > Options (Windows) or Mendeley Desktop > Preferences (MacOS), select the BibTeX tab.
 1. Ensure that Escape LaTeX special characters (#{}%& etc.) is checked.
 2. Ensure that Enable BibTeX syncing is checked
 3. Supply the path to where you want the bibliographic database to be located.
 - On Windows, the path this author used was C:\Users\myUsername\Documents\Bibliography
 - On MacOS, the path this author used was /Users/myUsername/Documents/Bibliography/
2. In order to use the CSS, in a web browser...
 1. Navigate to the following: <https://gist.github.com/dashed/6714393>
 2. Click on "Raw".
 3. Select all and copy
 4. In VSCode, select File > New File
 5. Paste the contents
 6. Using File > Save As, navigate to and create a folder in one's documents called CSS
 7. Save the file as: github-pandoc.css
3. In order to use the CSL, in a web browser...
 1. Navigate to the Official CSL Repository: [APA-5th-Edition](#)
 2. Select all and copy
 3. In VSCode, select File > New File

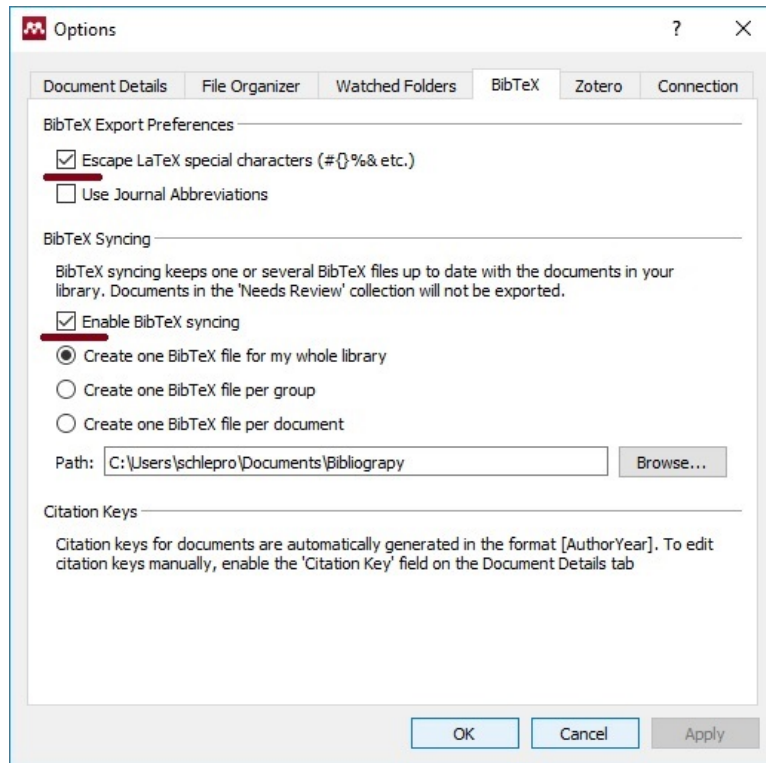


Figure 13: Mendeley BibTex options

4. Paste the contents
5. Using **File > Save As**, navigate to and create a folder in one's documents called **CSL**
6. Save the file as: `apa-5th-edition.csl`
4. Now to create the document...
 1. In VSCode, select **File > New File**.
 2. Select **File > Save As**
 3. Navigate to and create a folder in one's documents folder called **Markdown**.
 4. Call the file **Markdown.md**
 5. Add the following to the text in the file:

```

1 ---
2 bibliography: ../Bibliography/library.bib
3 csl: ../csl/apa-5th-edition.csl
4 css: ../css/github-pandoc.css
5 mode: self-contained
6 ---
7 [@Beier2002]
8
9 # References

```

Now to convert the Markdown file to .docx, PDF and HTML...

5. In VSCode's Explorer, right-click on Markdown.md
 - In **Windows** select **Open in Command Prompt**
 - In **MacOS** select **Open in Terminal**
6. A new pane will open up **in** VSCode, at the bottom of the main window. In this pane will be a command prompt that looks like either:

```
C:\Users\YourUsername\Documents\Markdown>
ComputersName:Markdown YourUsername$
```

7. Type `pandoc --filter pandoc-citeproc Markdown.md -o Markdown.pdf`, and press **Enter** to test PDF functionality.
8. Type `pandoc --filter pandoc-citeproc Markdown.md -o Markdown.html`, and press **Enter** to test HTML format output.
9. Type `pandoc --filter pandoc-citeproc Markdown.md -o Markdown.docx`, and press **Enter** to test the Microsoft docx format output.

```
1 $ pandoc --filter pandoc-citeproc Markdown.md -o Markdown.pdf
2 $ pandoc --filter pandoc-citeproc Markdown.md -o Markdown.html
3 $ pandoc --filter pandoc-citeproc Markdown.md -o Markdown.docx
4 $ ls
5 Markdown.md markdown.docx markdown.html markdown.pdf
6 $
```

10. Right click on Markdown.md in the **VSCode Explorer** near the top-left.
11. Select **Reveal in Explorer** (Windows) or **Reveal in Finder** (MacOS). This will open the `../Markdown` folder.
12. There should be four files. Open the docx, HTML and PDF documents to see what they look like.
13. The Markdown.md file, the one users are starting with, should look like [this](#)
14. The docx file should look like [this](#).
15. The PDF file should look like [this](#).
16. The HTML file should look like [this](#).

If one or all of the files did not get created, read the messages for clues on what went wrong. If a citation wasn't found, check the path to the bibliography, and check the key both in Mendeley, and in the Markdown.md file to make sure they are the same. If the styles did not render as expected, check the path to the CSL or CSS files. Verify where the files were stored.

When this author first ran the script, I encountered the error:

```
pandoc-citeproc: PandocResourceNotFound "../cs1/apa-5th-edition.csl"
```

This error indicated that the CSL file was not in the path specified in the Markdown.md file. It turned out that this author did not save to `/Users/myUserName/Documents/CSL/apa-5th-edition.csl`, but instead to `/Users/myUserName/Documents/Repo/Styles/apa-5th-edition.csl`. As a result, the line in the Markdown.md file needed to look like the following to work:

```
cs1: ../Repo/Styles/apa-5th-edition.csl
```

Once the relative path to the file pointed to the right place, the conversion completed without errors. As always, google is a fine resource to search for any error messages encountered.

Continuing with the Edu

From here on, it is assumed that readers have successfully created docx, PDF, and HTML files using Pandoc, and are interested in writing a file in Markdown to display VBA code blocks (or R or Python). Each section will be covered in some more detail than the minimal example.

New Projects in VSCode

Working with a new project in VSCode starts with adding a folder to the workspace. To do this use (on Windows) **File > Add Folder to Workspace** and select a folder or create a new one. Once this is finished, **File > Save Workspace As** and give it a shiny name, say ***Markdown***. This action sets the name of the project in VSCode.

Once the Workspace is created and saved, select **File > New File**. A tab will open up in the view-pane of VSCode called ***Untitled-1***. Before doing anything with the file, save it under a new filename by selecting: **File > Save As...** and save it in the workspace folder. Say for the current purposes that the file name is ***NewMarkdownFile.md***.

The File Extensions

There are many file types that VSCode can work with, and many more when the extensions for them are installed. The following file-extensions are the ones that this author has prepared users to code in so far using VSCode. **.md** is one file extension that indicates the file is a Markdown file. **R** is one extension for R code. **.py** is one extension for Python code. **.bas** is one extension for VBA that can be imported/exported to/from Excel. Saving the initial file using a known file extension is one way that VSCode and Pandoc figure out how to highlight a user's syntax.

Working with Pandoc-Markdown

This section includes a non all-inclusive list of the Markdown and Markdown code-block syntax that this author has found useful. Users are encouraged to reference the **Pandoc-Markdown** and **GitHub Flavored Markdown** documentation. In addition, this author found a **similar site** useful, along with Google searches for Markdown tasks, such as “Markdown change color”. Included topics are Markdown headings, two HTML tags: color and anchor; links, inline code, block code, bulleted and numbered lists, and images are covered.

Markdown Headings

Except for when using a YAML header (discussed later), the Markdown document usually starts with a first level heading.

```
20 # Heading 1 {#Identifier}
21
22 New paragraph.
23
24 ## Heading 2 {#SecondID}
25
26 ...
27
28 ##### Heading 6 #####
29
30 [return link to #Identifier](#Identifier "the first ID above.")
```

Preceding pound signs (up to 6) delineate headings. Pound signs following the heading text are optional. A space after the pound sign is required, and a new line before and after the heading is recommended by the documentation, and enforced when using Markdown **linting**. The above headings render as follows:

Heading 1

New Paragraph.

Heading 2

...

Heading 6

[return link to #Identifier](#)

Working with Markdown

Support for HTML Tags

HTML tags to modify, for example, text color when using a markdown renderer such as Pandoc or VSCode preview, can be nested in headings.

```
397 ##### <span style="color:blue">Heading 4</span> #####  
398
```

Results in:

Heading 4

Additionally, **named anchors** can be created as well and linked to from other places in the document. ``, for example, creates the named anchor `ThisText`, which can be linked to in other places in the document by using link syntax: `[Goto ThisText](#ThisText "Going to ThisText")`.

The current document has level one and level two headings also defined with anchors. The following embeds the color style inside a heading demarcation, and adds an anchor for the heading at the end.

```
411 #### <span style="color:green">Heading 2 with Color and an ID</span><a id="H2"></a>  
412
```

Renders as:

Heading 2 with Color and an ID

Links

Links are specified in Markdown as follows:

```
425 [inline link text](actual-url-or-anchor "mouse-over text")  
426
```

The structure of links has been utilized extensively in this document. For example `[Google](www.google.com "Goto google")` results in [Google](#).

Links can be used to go to anchors in the document as well. To go back to the green Heading 2 above, just use the following code `[goto heading 2](#H2 "going to heading 2")` results in the following: [goto heading 2](#).

Inline code

Inline code is delineated between back-ticks, or `“”`. Thus `‘some code’` results in `some code` when the text is rendered. Generally, since the type of code is not indicated when using inline code, there will not be any highlighting.

Block Code

Block code is delineated between:

1. At least 4 spaces at the beginning of a line, which, just like headings, is preceeded and followed by a newline.
2. Three back-ticks on newlines, preceeded and followed by a newline, where the first set of back-ticks is followed by a text qualifier indicating what language to use for syntax highlighting.

Thus, at least four spaces (below, the spaces are indicated by the preceeding underline) before the code on each line will result in a code block (if the block is separated from other paragraphs by a newline:

_____#### Heading 4 ####

_____Here's a paragraph.

Renders to the following:

Heading 4

Here's a paragraph.

Without highlighting, and:

```
```{#Heading4b .markdown .numberLines startFrom="461"}
```

#### Heading 4 ####

Here's a paragraph.

```

Renders to the following code block:

```
461
462 #### Heading 4 ####
463
464 Here's a paragraph.
```

with highlighting.

When using the back-ticks, `markdown`, `r`, `py`, `vb` indicate markdown code, r-code, python code and VBA, respectively.

Lists

Lists use simple syntax as well. Just precede each line by one of the **valid bullets**, **+**, *****, or **-** and use tabs to set the list level.

```
487 + Hello world
488 - Hello again
489   * And once more, hello
490
```

looks like:

- Hello world
- Hello again
 - And once more, hello

Numbered lists are the same as bulleted lists, just preceded by the number 1, or any common delimiter.

```
52 1. First item
53 1. 2nd item
54 1. 3rd item
55
```

looks like:

1. First item
2. 2nd item
3. 3rd item

Images

Images are also straight-forward in Markdown. The syntax is just like a link, preceded on the same line with a **!**.

```
521 ![Caption-text](url-or-file-path "mouse-over-text"){ width=300px }
522
```

See [the Pandoc manual](#) to see specifics on when images will be treated like figures, and when they will not.

Citations

Citations are handled by using `[@key]` convention, where **key** is a unique qualifier from one's Mendeley database wherever one needs the citation to be placed. Well, actually, the identifier comes from the local BibTex database, which Mendeley or Zotero can help one setup. If one needs to suppress the author, just subtract `-` before the `@`, as in `[-@key]`. Multiple keys in one citation are separated with a semicolon `;` within the same set of brackets. Prefix and

postfix text is simply typed in within the brackets. When converting from Markdown to one's output formats, the option `--filter pandoc-citeproc` will be needed in the command line.

So, citing the article is as easy as pie [`@Tellinghuisen2000`]. Alas, the same author [`-@Tellinghuisen2000`] concluded something cool as well.

Renders to:

So, citing the article is as easy as pie (Tellinghuisen, 2000). Alas, the same author (2000) concluded something cool as well.

Mendeley Prereq

In order to use citations, Pandoc will need to know where to go to get one's bibliography. Mendeley Desktop synchronizes with one's local BiBTeX database so one's bibliography on the local machine is always up to date.

In Mendeley, under **Tools** select **Options**, then select the **BibTeX** tab. Verify that **Enable BibTeX Synching** is selected. Check and/or set the path to the **Library.bib** file. Only the folder will be shown in the path (not the file **library.bib**), which may look like the following:

```
> /Users/YourUsername/Documents/Bibliography
```

Metadata

At the top of a markdown file, users can add what's called the "YAML Meta-Data" (aka a standard for configuration files; Ben-Kiki, Evans, & Net, 2009) with newlines with 3 dashes each, as follows:

```
1 ---
2 bibliography: ../Bibliography/library.bib
3 css: ../CSS/GitHub-pandoc.css
4 csl: ../CSL/apa-5th-edition.csl
5 mode: self-contained
6 title: "All about the markdowns baby"
7 subtitle: "Installing VSCode, Mendeley, MiKTeX, Pandoc, and Git"
8 date: "18 February 2018"
9 author: "Keith Chamberlain"
10 company: "www.ChamberlainStatistics.com"
11 abstract: |
12     Optionally, the abstract can start on the second line after |
13     the title "abstract:" and when using the '|' (pipe) symbol. |
14     It is okay for the abstract to go across lines. |
15     \pagebreak
16 ---
```

- **bibliography:** `path` indicates where Pandoc should look to find the BiBTeX bibliography. Mendeley's bibliography works out of the box if the option to synchronize to a local BiBTeX library is set.
- **css:** `path` indicates where Pandoc should look to find the file describing the document's style rules for the HTML output.
- **cs1:** `path` indicates where Pandoc should look to find the file describing the citation style rules.
- **mode:** `self-contained` says to store images as part of the file rather than linking to them and having the reader load the images separately.
- **title:**, **subtitle:**, **date:**, **author:**, **company:** all self explanatory.

Bibliography, css, cs1 & self-contained have command line equivalents for Pandoc. For more on YAML meta-data, see the [YAML website](#).

nocite

Say one wants to include citations in a bibliography or references section, even though they were not referred to in the main document. This is straight forward with Pandoc-Markdown.

```
581 ---
582 nocite: |
583     @key1, @key2, ...
584 ---
```

Then, when converting from Markdown to, say, docx, and assuming the file is called marky.md, the command line call would include the `--filter pandoc-citeproc`, as in:

```
> pandoc --filter pandoc-citeproc marky.md -o marky.docx
```

And that's pretty much it! See the [manual](#) for more information.

Markdown Summary

There were many topics covered when talking about Pandoc-Markdown and citations: the [installation of software](#) was covered; namely, VSCode, MiKTeX, Pandoc Git, GitHub Desktop and Mendeley. A series of extension packages was installed for LaTeX and VSCode to make Pandoc and VSCode fully operational with PDFs and the programming language of one's choice.

Pandoc was chosen to handle bibliographies in file conversion, while Mendeley managed the local bibliographic library and synchronized with the one on the cloud. PATHs were set so the different softwares could find each other's binaries. Then several Pandoc-Markdown examples were covered that expose certain aspects of the language. Finally, citations and metadata were discussed. Now it is time to play with some code...

The VBA code That Was Promised

As stated earlier in this post, this author was not planning on using VBA for anything more than interfacing Python to Excel. Curiosity and ease of use for Chemists to incorporate VBA into their own workflow, without having to have Python installed, prevailed, however, so some more VBA work was done.

The following code is about being able to calculate the difference between two values, and if the appropriate form of the function is called, then restrict the values to positive numbers and report that difference in PPM. Such a set of functions may find use in a mass PPM difference calculator in the Chemical and Physical sciences, for example.

diff()

```

646 ' The lower level and more general difference function. Minimal
647 ' error checking. The doubles are set to 'nearly zero' by default
648 ' in order to prevent divide by zero errors.
649 Private Function diff(Optional observed As Double = 1E-16, _
650                       Optional expected As Double = 1E-16, _
651                       Optional fitOption As Long = 2, _
652                       Optional reference As Double = 1E-16) _
653                       As Variant
654     ' Declarations
655     Dim setit As Long
656     setit = 1
657     ' Case
658     Select Case fitOption
659         Case 1 ' Result * 100 = Classic % difference
660             ' (e.g. normalized by the mean)
661             If (WorksheetFunction.Average(observed, expected) _
662                 <> 0) Then
663                 diff = (observed - expected) / _
664                     WorksheetFunction.Average(observed, _
665                         expected)
666             Else: diff = (observed - expected) / 1E-16
667             End If
668         Case 2 ' Classic "mass" difference (e.g. normalized
669             ' by expected)
670             diff = (observed - expected) / expected
671             ' Log difference
672         Case 3 ' Log PPM difference, similar to classic
673             diff = (Log(observed) - Log(expected))
674         Case 4 ' Chamberlain difference

```

```

675         diff = ((observed - expected) / reference)
676     Case 5 ' Log Chamberlain PPM difference
677         If (observed - expected) < 0 Then setit = -1
678         diff = setit * (Log(reference + Abs(observed - _
679             expected)) - Log(reference))
680     Case Else
681         diff = "Option Out of Range" ' = CVErr(xlErrNA)
682         ' Using a more sensible error instead. Requires that
683         ' the function is a Variant to return strings AND
684         ' doubles.
685     End Select
686 End Function

```

The above function calculates one of 5 differences, depending on the parameters in the calling environment. Case 1 results in a classic percent difference, where the difference value is normalized by the mean of the observed and expected values.

$\text{difference} / \text{mean}(\text{observed}, \text{expected}) = 2 * \text{difference} / (\text{observed} + \text{expected})$

Case 2 results in a “Classic mass difference” with the known form difference/expected rather than dividing by the average of observed and expected.

$\text{difference} / \text{expected}$

3 calculates the Log PPM difference, which results in values very close to Classic mass differences, but by \log_{10} arithmetic instead.

$\text{Log}(\text{observed}) - \text{Log}(\text{expected})$

Case 4 and 5 are what this author proposed as the Chamberlain PPM Difference, and Log Chamberlain PPM difference, otherwise known as **Scaled PPM Differences**. As mass increases, the difference (a ratio) of one electron (a constant) gets smaller (divided by a larger and larger number). The unscaled PPM difference of that electron at 200u is very different than the unscaled PPM difference of that same electron at a mass of 30,000u. In order to control for that scaling, the divisor in the PPM difference calculation can be set such that regardless of the mass, the difference of an electron means the same thing. Perhaps that value for the electron difference is set to be equal to 10 ppm irrespective of whether the mass is 200u, or 30,000u.

Case 4 is just using a divisor that is set a-priori:

$\text{difference} / \text{reference}$

Case 5 looks at the absolute log difference, and then subtracts $\log(\text{reference})$. Negative differences are handled by a check before the calculation and the negative sign, if exists, is added back on at the end.

$\text{Log}(\text{reference} + |\text{difference}|) - \text{Log}(\text{reference})$

classicPercentDiff()

```

713 ' Function to return the classical percent difference between
714 ' the observed and expected values after conditioning the input
715 ' for diff() (a wrapper to diff()) for option 1 of diff() only,
716 ' that also handles negative values.
717 Public Function classicPercentDiff(Optional value1 As Double _
718                                   = 1E-16, Optional value2 As Double _
719                                   = 1E-16) As Variant
720     ' Error handling
721     If value1 = 0 Then value1 = 1E-16
722     If value2 = 0 Then value2 = 1E-16
723     ' Calculator call
724     classicPercentDiff = diff(value1, value2, 1) * 100
725     If classicPercentDiff = -0 Then classicPercentDiff = 0
726     ' To clean up the -0 condition from 0/(-1).
727 End Function

```

classicPercentDifference() is simply a convenience wrapper for **diff()** that checks some of the input, makes some guesses about what the user wants and uses those guesses to set some defaults, and cleans up some side-effects that were discovered. ($0/(-1)$), for example, resulted in -0 and caused condition checking to fail, so it had to be handled explicitly.

massDiff()

```

734 ' Function to return the PPM mass difference between the
735 ' observed and theoretical masses after conditioning the
736 ' input for diff() (a wrapper to diff()) to be specific
737 ' for masses
738 Public Function massDiff(Optional observed_Mass As Double = _
739                          1E-16, _
740                          Optional exact_Mass As Double = 1E-16, _
741                          Optional fitOption As Long = 2, _
742                          Optional reference_Mass As Double _
743                          = 200#) As Variant
744     ' Error handling
745     If observed_Mass <= 0 Then observed_Mass = 1E-16
746     If exact_Mass <= 0 Then exact_Mass = 1E-16
747     If reference_Mass <= 0 Then reference_Mass = 1E-16
748     If fitOption <= 2 Then fitOption = 2
749     If fitOption >= 5 Then fitOption = 5
750     ' Calculator call
751     massDiff = diff(observed_Mass, exact_Mass, fitOption, _

```

```
752         reference_Mass)
753     If (massDiff = "Option Out of Range") = False Then _
754         massDiff = massDiff * 1000000
755     If massDiff = -0 Then massDiff = 0
756     ' To clean up the -0 condition from 0/(-1).
757 End Function
```

massDiff() is a convenience wrapper for **diff()** that does some error checking, makes reasonable assumptions about what the user wants by setting defaults, and makes sure the input masses are positive. The awkward condition of returning -0 is handled just as before, as it was encountered here as well.

refDiff()

```
764 ' refDiff() provides a way to back-calculate what the reference
765 ' would need to be to result in a given target-difference,
766 ' using one method for the calculation.
767 Public Function refDiff(Optional difference As Double = 1E-16, _
768     Optional target As Double = 0.000001) _
769     As Variant
770     refDiff = target / difference
771 End Function
```

Much like **diff()**, **refDiff()** does minimal to no error checking. Convenience wrappers could easily be written to ensure inputs are within a certain range, for example, just as the wrappers for **diff()**. To use the function, supply the desired difference to set a reference for, say, the mass of an electron = 0.00054857990924 (Beier et al., 2002), and one wants to call that difference 10 ppm, then target = 10E-6. The reference is 54.85799u. If one wants to normalize the mass of an electron to be 1 ppm, then the reference mass is 548.5799u. This author believes using scaled references will enable setting the same threshold value for mass conformity across the whole mass range.

Now readers have also seen some VBA code with syntax highlighting, which overcomes this authors first, and very feeble attempt to code VBA blocks in R Markdown when Visual Basic highlighting is not yet supported there (as of this writing). The VBA code for the mass functions is located on this author's [GitHub site](#).

References

- Beier, T., Häffner, H., Kluge, H. J., Quint, W., Verdú, J., Häffner, H., Hermanspahn, N., et al. (2002). New Determination of the Electron's Mass. *Physical Review Letters*, 88(1), 4.
- Ben-Kiki, O., Evans, C., & Net, I. dot. (2009). YAML Aint Markup Language. Retrieved from <http://www.yaml.org/spec/1.2/spec.html>
- Cascading Style Sheets Working Group. (2018). *Cascading Style Sheets*. Retrieved from <https://www.w3.org/Style/CSS/members>
- Chacon, S., & Straub, B. (2014). *Pro Git* (2nd ed., p. 486). New York: Springer Science+Business Media. Retrieved from <https://git-scm.com/book/en/v2>
- Github Inc. (2018). GitHub Desktop. Retrieved from <https://desktop.github.com/>
<https://help.github.com/desktop/guides/getting-started-with-github-desktop/>
- MacFarlane, J. (2018). *Pandoc: A universal document converter*. Retrieved from <http://pandoc.org/MANUAL.html>
- Microsoft. (2016). Visual Basic for Applications (VBA). Retrieved from <https://msdn.microsoft.com/en-us/vba/language-reference-vba/articles/visual-basic-language-reference>
- Microsoft. (2018). Visual Studio Code. Retrieved from <https://code.visualstudio.com/>
- R Core Team. (2017). *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from <https://www.r-project.org/>
- RStudio Team. (2015). R Markdown Reference. Retrieved from <https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf>
- RStudio Team. (2016a). R Markdown Cheat Sheet, 5, 1–2. Retrieved from <https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf>
- RStudio Team. (2016b). *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, Inc. Retrieved from <http://www.rstudio.com/>
- Schenk, C. (2018). MiKTeX. Retrieved from <https://miktex.org/>
- Tellinghuisen, J. (2000). A simple, all-purpose nonlinear algorithm for univariate calibration. *Analyst*, 125(6), 1045–1048. The Royal Society of Chemistry. Retrieved from <http://dx.doi.org/10.1039/B002171G>
- W3schools.com. (2018). CSS Tutorial. Retrieved from <https://www.w3schools.com/css/default.asp>
- Zelle, R. M., Bennett, F. G. J., & D’Arcus, B. (2015). Citation Style Language Specification. Retrieved from <http://docs.citationstyles.org/en/stable/specification.html>
- Zoomer Analytics LLC. (2018, January). xlwings: Python for Excel. Free & Open Source. Retrieved from <http://docs.xlwings.org/en/stable/quickstart.html>