An Extended Introduction to LaTeX

Mayya Komisarchik*

September 30, 2016

What is LaTeX?

LaTeX is a markup language, not a programming language. This makes LaTeX a lot more like html than $\mathbb{R}$ or Python. That means that the internal logic of LaTeX is much simpler than what you see for any of the other programming languages you've learned. All you need to do is make sure that everything is wrapped in the correct "tags" that identify every style and environment you'd like your math or text to appear in. That's really it!

The Preamble

Let's literally start at the top. The top of the code for each of your LaTeX files should feature a preamble with the following:

- The definition of your document class
- Calls to all of the packages you are going to be running
- Any new commands you want to define in LaTeX

Let's go through how to structure all of these arguments in some detail.

*See https://tobi.oetiker.ch/lshort/lshort.pdf for an even more extended version of the information I'm presenting here; this document is based heavily on Oetiker, Partl, Hyna and Schegl's "The Not So Short Introduction to LaTeX 2."

Document Classes

Document classes in LaTeX specify the type of document that your code will create. LaTeX supports a wide variety of document classes, and there are downloadable extensions to LaTeX that allow you to expand the universe of document classes available to you. The list of document classes below is far from exhaustive, but it itemizes the document classes in LaTeX that you'll use most frequently in your academic career:

Table 1: Basic Document Classes in LaTeX

Document Class	Short Description
`article`	academic articles, papers, expository documents
`book`	books.
`letter`	letters.
`beamer`	presentations.
`slides`	makes slides: default format set to large sans serif font.
`report`	reports containing multiple chapters: thesis, short books
`minimal`	sparse formatting class; just sets page size and base font
`proc`	record of proceedings, derived from article class

Defining a Document Class

The general syntax for defining your document class is as follows:

\documentclass[*options*]{*class*}

The options field lets you customize the appearance of your document. Options include things like font size, the assumed paper size, whether or not there is a titlepage, whether the document is presented in portrait or landscape, etc. Here are some basic option definitions in available for document classes in LaTeX:
When you invoke these options, you'll use the syntax at the beginning of this section with your comma-separated options in square brackets. For instance:

```
\documentclass[12pt, legalpaper, titlepage]{article}
```

Try setting that document class for this template!

Table 2: Options for Document Classes in LaTeX

Option	Short Description
`10pt`, `11pt`, and `12pt`	determine your font size. Default is 10pt
`letterpaper`, `legalpaper`	determines paper size. Default is letterpaper.
`fleqn`	typesets formulae left aligned instead of centered.
`leqno`	numbers equations on the left instead of the right.
`titlepage, notitilepage`	specifies whether titlepage gets own page or not.
`landscape`	sets document in landscape mode
`handout`	makes printable versions of beamer slides

Packages

Much like $\mathbb{R}$, LaTeX has packages (bundles of code) that add extended functionality to base LaTeX code. Also much like with $\mathbb{R}$, there are far too many packages written for LaTeX to discuss them all in detail, but the subsections below will review the functions in the most common. Packages are "called" at the beginning of your LaTeX document using the command:

\usepackage[*options*]{*package*}

amsmath, amssymb

Documentation: `ftp://ftp.ams.org/pub/tex/doc/amsmath/amsldoc.pdf`

Overview: Both of these packages are part of the $\mathcal{AMS}$-LaTeX bundle written by the *American Mathematical Society*. This bundle is part of every recent LaTeX distribution you can download today, so you don't need to install anything in addition to LaTeX in order to use it.

The `amsmath` package allows you to use equation, align, gather, multline, split and other math environments to write all of your equations. We'll discuss these in more detail throughout the section on math environments. The `amssymb` package allows you to draw many of the math symbols you use daily in your problem sets and equations. There are numerous resources online that will help you find the commands that generate particular math symbols. Here is a particularly useful one for the symbols available in base LaTeX math mode: `http://www.auburn.edu/~tamtiny/Symbols.pdf`, and one detailing symbols available through `amssymb`: `http://milde.users.sourceforge.net/LUCR/Math/mathpackages/amssymb-symbols.pdf`. You'll generally want to run both of these in any document that contains math.

graphicx

Documentation: `http://mirrors.rit.edu/CTAN/macros/latex/required/graphics/grfguide.pdf`

Overview: This is the package that allows you to import and manage graphics in LaTeX. This includes the command `\includegraphics`. You'll always want to run this.

color

Documentation: `http://ctan.sharelatex.com/tex-archive/macros/latex/required/graphics/color.pdf`

Overview: This package implements LaTeX's ability to read colors. This package allows LaTeXto compile the following commands:

- `\textcolor{declared-color}{text}` (changes in-line text color)
- `{\color{declared-color}some text}` (changes in-line text color)
- `\pagecolor{declared-color}` (changes background color on your pages)
- `\colorbox{declared-color}{some text}` (changes background color behind a particular block of text)

By default, LaTeX recognizes the following pre-defined colors by name: black, blue, brown, cyan, darkgray, gray, green, lightgray, lime, magenta, olive, orange, pink, purple, red, teal, violet, white, and yellow. You may want to use an expanded library of 68 colors available in dvips (see `https://en.wikibooks.org/wiki/LaTeX/Colors#Adding_the_color_package`). To do that, when you initialize the color package, instead of running:

```
\usepackage{color}
```

you'll want to initialize the following:

```
\usepackage[dvipsnames]{xcolor}
```

That allows you to use fancy colors like cornflower blue, **sepia**, and red orange.

hyperref

Documentation: `http://ctan.math.utah.edu/ctan/tex-archive/macros/latex/contrib/hyperref/doc/manual.pdf`

Overview: This package extends the functionality of cross-referencing commands in LaTeX. It allows you to generate hyperlinks that navigate between sections and items in your document, and it also lets you generate links to urls in your code. This package is the reason I can include working urls in this document with the `\url{}` command! Hyperref includes commands that generate dynamic references to figures and tables; we'll discuss this further when we get to figures, tables, floats and labels.

setspace

Documentation: `https://en.wikibooks.org/wiki/LaTeX/Paragraph_Formatting`

Overview: `setspace` lets you change the line spacing in a document. It provides the commands:

- `\doublespace`
- `\onehalfspace`
- `\singlespace`
- `\begin{spacing}{baselinestretch}` ... `\end{spacing}`

These commands let you customize your paragraph spacing. Each of the first three commands above can be used anywhere in your document, and you can switch your paragraph spacing throughout your document. The last command lets you define a completely custom paragraph spacing. See the documentation on this package for the syntax that will help you set your own spacing limits.

cancel

Documentation: `http://ctan.mackichan.com/macros/latex/contrib/cancel/cancel.pdf`

Overview: This package just lets you wrap math or text in the tag `\cancel{}` in order to strike it out. Available commands in this package include:

- `\cancel` (slash through)
- `\bcancel` (backslash through)
- `\xcancel` (draws an x through)
- `\cancelto` (draws a diagonal arrown through the expression that points to the value it becomes)

This package essentially lets you turn $\frac{2}{2\pi}$ into $\frac{\cancel{2}}{\cancel{2}\pi}$ or $\frac{\bcancel{2}}{\bcancel{2}\pi}$ or $\frac{\xcancel{2}}{\xcancel{2}\pi}$ or $\frac{\cancelto{1}{2}}{\cancelto{1}{2}\pi}$

fullpage

Documentation: `http://mirrors.ibiblio.org/CTAN/macros/latex/contrib/preprint/fullpage.pdf`

Overview: This package simply sets the margins of your document to take up the full page. You'll notice that the default margins in an article document class are much wider than the ones in this document. Don't believe me? Just comment out the fullpage package call in the preamble to this document. This package adapts the fullpage setting to the paper type you identify in your document class options. Options for this package include:

- `in` (sets margins to one inch)
- `cm` (sets margins to 1.5 cm)
- `plain` (this is the default, it sets the plain page style with footers but no headers)
- `empty` (no headers and no footers)
- `headings` (headers and footers)

listings

Documentation: `http://mirrors.ibiblio.org/CTAN/macros/latex/contrib/listings/listings.pdf`

Overview: This package is designed to let you print source code from other languages into LaTeX, just as that code might appear in the appropriate language. You can copy your code by using a listing environment, for instance:

```
\begin{lstlisting}
some ℝ code here
\end{lstlisting}
```

Or you can use the `\lstinputlisting{filename.R}` to import an entire block of code. We'll talk more about this in the section on importing ℝ code into your writeups. You can set the appearance of your source code usting the `\lstset{}` command in the preamble of this document.

tikz

Documentation: `http://cremeronline.com/LaTeX/minimaltikz.pdf`

Overview: Ti*k*Z lets you draw your own graphics and illustrations into a LaTeX document! It introduces the tikzpicture environment, in which you can draw some of the figures you've seen me present to you in Gov 2000 problem sets and section slides. The general syntax for a Ti*k*Z environment is:

```
\begin{tikzpicture}
your image code
\end{tikzpicture}
```

Note that you can wrap Ti*k*Z pictures in figure environments the same way that you do with other graphics. This lets them "float" the way that imported graphics do.

framed

Documentation: `http://mirror.utexas.edu/ctan/macros/latex/contrib/framed/framed.pdf`

Overview: This package legit just draws boxes around things. Anything you wrap in the tags: `\begin{framed}` and `\end{framed}` will appear in a box. Here's the general syntax:

```
\begin{framed}
\begin{center}
I'm in a box!
\end{center}
\end{framed}
```

Gives:

I'm in a box!

If you define a color usig the commands `\definecolor{shadecolor}{rgb}{r, g, b}` or `\colorlet{shadecolor}{color}`, you can use the tags textttbegin`{shaded}` and `\end{shaded}` to draw boxes with a background color. Check this out, running the following:

```
\colorlet{shadecolor}{gray!40}
\begin{shaded}
\begin{center}
I'm in a box!
\end{center}
\end{shaded}
```

produces:

I'm in a box!

This is a handy way to emphasize answers and to set your $\mathbb{R}$ code apart in your writeup since you can wrap the environments that let you print source code in frame and shade boxes. We'll talk about that later in this workshop.

Custom Commands

LaTeX allows you to define your own commands. Defining your own commands can significantly shorten the amount of code you have to write in order to typeset particular expressions;

this takes a little bit of investment up front, but once you've defined a set of commands you intend to use consistently you can just copy them into new documents or your base template and save a ton of time.

The general syntax for defining a new command in LaTeX looks like this:

`\newcommand{name}[num]{definition}`

Above, `name` refers to the name you want to give your command - *including* the leading "\". The optional `num` field specifies the number of arguments your command can take, with an upper limit of 9. And the definition in braces represents the code you want LaTeX to run when you apply your new command. To refer to your arguments in the definition, use the hash symbol # and the number of the argument.

I've put a few custom commands in this template for you. Let's walk through one of them to look at a specific command definition and see why it's helpful. Notice that the preamble of this document contains the following command:

`\newcommand{\pd}[2]{\frac{\partial#1}{\partial#2}}`

In LaTeX, if you wanted to write out the formula for a partial derivative you'd use the following code:

`\frac{\partial}{\partial}`

with all of the math symbols you want to include in your derivative to the right of each `partial` symbol. But that's kind of a giant pain the a##. So, in the command defined in this example, we tell LaTeX to initialize a new command, which runs if you type `\pd{}{}`, takes 2 arguments (for the variables with respect to which you're differentiating), and generates the same form of the partial derivative. Try writing the code for the following partial derivative yourself using the `\pd{}{}` command:

$$f(x,y) = x^2 + 6y^3 + y^2 + 14$$
$$\frac{\partial f(x,y)}{\partial y} = 18y^2 + 2y$$

Much shorter than using the full partial derivative syntax!

Math Environments

This section covers various types of math environments you're likely to use on your problem sets, with some discussion of the benefits and drawbacks of each.

The simplest way to write math in a LaTeX document is to write in-line math. That just means you need to wrap your mathematical expressions in $ operators so that LaTeX reads them as mathematical symbols. If you choose to do this, you'll just format all of the breaks and spaces between your expressions manually - the way that you might do with text. But that's annoying! You might want to explore some math environments that do things like center, number, and align math expressions for you.

Equation Environments

The equation environment is one of the math environments in the $\mathcal{AMS}$-LaTeX bundle. You can initialize an equation environment in several ways. For numbered equations, you just need to wrap your equations in `\begin{equation}` and `\end{equation}` tags. You'll get something like this:

```
\begin{equation}
    x = 3 + y
\end{equation}

\begin{equation}
    y = 7 - x
\end{equation}
```

which gives:

$$x = 3 + y \tag{1}$$

$$y = 7 - x \tag{2}$$

Each equation environment can only house a single equation, so you need a new equation environment for each separate equation you need to write. If you don't want to number your equations, just add a star, *, to your equation command like this:

```
\begin{equation*}
    x = 3 + y
\end{equation}

\begin{equation}
    y = 7 - x
\end{equation*}
```

Alternatively, you can write an equation in-line with the rest of your text if you set it off with two $ operators as follows:

```
$$x = 3 + y$$
$$y = 7 - x$$
```

For single equations that are too long to fit on a single line, you can use the `multline` environment. This environment lets you use the \\ operator to force all the subsequent math onto a new line just as you would with text. Here's an example:

```
\begin{multline}
a + b + c + d + e + f + g + h + i + j + k + l + m + n + o +
p + q + r + s + t + u + v \\ = w + x + y + z
\end{multline}
```

Gives:

$$a+b+c+d+e+f+g+h+i+j+k+l+m+n+o+p+q+r+s+t+u+v \\ = w+x+y+z \quad (3)$$

Notice that this is a pretty space-hungry approach to writing math. There's a great deal of space between equations. You may also, especially when you're doing proofs or writing out the steps you took to solve some algebraic problem, want to write a series of related math expressions in a single chunk. If that's your aim, you're much better off using an align or an eqnarray environment.

Align Environments

Align environments have two advantages over standard equation environments: (1) they let you stack multiple equations in a single environment and (2) they let you choose the character in each line where you'd like to left-align your equations. In an align environment, you can use the & operator to align your math expressions and the \\operator to start a new equation. Here's how it works:

```
\begin{align*}
f(x, y)          & = x^2 + 3x + 4y^3 + y^2 + 9y \\
\pd{f(x, y)}{y} & = \pd{[x^2 + 3x + 4y^3 + y^2 + 9y]}{y} \\
                 & = 12y^2 + 2y + 9 \\
\end{align*}
```

Gives:

$$\begin{aligned} f(x,y) &= x^2 + 3x + 4y^3 + y^2 + 9y \\ \frac{\partial f(x,y)}{\partial y} &= \frac{\partial [x^2 + 3x + 4y^3 + y^2 + 9y]}{\partial y} \\ &= 12y^2 + 2y + 9 \end{aligned}$$

Notice that I've placed the & operator before the = sign since I want all of my equations to align at =. Notice also that I don't actually need to have anything to the left of my & operator in order to align text.

Equation Arrays

You can also use the equation array environment to write a series of equations. This environment is very similar to the align environment. You're still using the \\to begin a new equation, but if you want to align equations or pieces of a single equation that's running onto another ling you need to *wrap* your alignment character in & operators. Here's how it works:

```
\begin{eqnarray*}
f(x, y)          & = & x^2 + 3x + 4y^3 + y^2 + 9y \\
\pd{f(x, y)}{y} & = & \pd{[x^2 + 3x + 4y^3 + y^2 + 9y]}{y} \\
                 & = & 12y^2 + 2y + 9 \\
\end{eqnarray*}
```

gives:

$$\begin{aligned} f(x,y) &= x^2 + 3x + 4y^3 + y^2 + 9y \\ \frac{\partial f(x,y)}{\partial y} &= \frac{\partial [x^2 + 3x + 4y^3 + y^2 + 9y]}{\partial y} \\ &= 12y^2 + 2y + 9 \end{aligned}$$

This environment has some spacing advantages over align. For instance, notice what align does by default with a single equation that's too long to fit on a single line:

```
\begin{align*}
f(x) &= a + b + c + d + e + f + g + h + i + j + k + l +
        m + n + o + p + q + r + s + t + u + \\
     & v + w + x + y + z
\end{align*}
```

produces:

$$f(x) = a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + r + s + t + u + v + w + x + y + z$$

So this is a problem. I've aligned to my equal sign, which is in fact what I want, but when I roll over to the next line I can only align before the first term, v. But that aligns v to my equal sign, when I clearly want v to come after my equal sign. There are some commands that would force some negative space before v on my second line here, but those are annoying and require lots of extra syntax and typing. Literally, why? I can avoid this easily with the equation array environment:

```
\begin{eqnarray*}
f(x) &=& a + b + c + d + e + f + g + h + i + j + k + l + m
         + n + o + p + q + r + s + t + u + \\
     && v + w + x + y + z
\end{eqnarray*}
```

produces:

$$\begin{aligned} f(x) &= a + b + c + d + e + f + g + h + i + j + k + l + m + n + o + p + q + r + s + t + u + \\ &\quad v + w + x + y + z \end{aligned}$$

Aligning in the equation array environment avoids this issue automatically.

Arrays of Equations

You can typeset equation arrays, such as the arrays we use to express PMFs in Gov 2000, using the array environment. The first thing to point out about arrays is that they are not, in themselves, math environments. To get LaTeX to recognize the mathematical syntax you enter into an array, you'll need to wrap your array in some other type of math environment (eqnarray, align, equation, etc.)

Arrays function a bit like tables. To use an array environment, you'll need to initialize an array using the `\begin{array}` tag. Much like you would in a table, you'll want to tell LaTeX how many columns you want and how to align them. You'd do that by adding an alignment tag right after `\begin{array}`. So, for example, if I wanted an array with 3 columns that were all center-aligned I'd write `\begin{array}{ccc}`. You can create column separators in an array using the | characters between your alignment indicators just like you would in a table. So if I wanted a line between the first column and the remaining columns in my array I'd just set my alignment tag to `{c|cc}`. The command `\hline` creates horizontal lines between rows in an array. Braces, parentheses, brackets, and other symbols you might want surrounding your array can be generated using `\left` and `\right` in conjunction with standard math symbols. The `\left` and `\right` automatically size symbols to fit the object they wrap.

Here's an example. The following code:

```
\begin{equation*}
f_{X}(x) = \left\{
           \begin{array}{rl}
              \frac{1}{b - a} & \text{if } a \le x \le b,\\
               0 & \text{ otherwise}
           \end{array} \right.
\end{equation*}
```

produces

$$f_X(x) = \left\{ \begin{array}{rl} \frac{1}{b-a} & \text{if } a \le x \le b, \\ 0 & \text{otherwise} \end{array} \right.$$

A few things to note about this. First, my array is inside of an equation environment so that LaTeX can read the math. Second, I'm creating my left brace using `\left{` and `\right.`, where the period indicates that no symbol should be printed with the right tag. LaTeX will break if you don't always have a right tag to close your left tag. Finally, just as I would in a table, I'm separating my actual math into columns using the & operator. It's that easy!

Tables in LaTeX

This section covers basic table creation and formatting in LaTeX . There is a lot more you can do with LaTeX tables than what you'll see in this document, and there are packages currently available for LaTeX that greatly enhance your ability to format tables.

Basic Tabular Environments

Tables are generally defined in a `tabular` environment, so the contents of each table you make should generally be wrapped in `\begin{tabular}` and `\end{tabular}` tags. Just as you might with an array, you can use pipes in combination with `l, c, r` characters to specify the column alignment in your tabular environment. Here's an example you might recognize from a recent problem set:

```
\begin{center}
\begin{tabular}{ c|c }
    Y $\downarrow$ & X = 1 \\
    \hline
    1 & $\alpha P(Y_g = 1)$ \\
    0 & $\beta (1-P(Y_g = 1))$\\
    \hline
    $P(X_g = 1)$ & $\alpha P(Y_g = 1) + \beta (1-P(Y_g = 1))$\\
\end{tabular}
\end{center}
```

Yields:

Y $\downarrow$	X = 1
1	$\alpha P(Y_g = 1)$
0	$\beta(1 - P(Y_g = 1))$
$P(X_g = 1)$	$\alpha P(Y_g = 1) + \beta(1 - P(Y_g = 1))$

Multicolumn and Multirow Labels

Suppose I wanted to add a column label that stretched across both columns for X and Y. Maybe I wanted to make it clear that they were indicators of undergraduate voting patterns. I could do that using the `\multicolumn` command. The `\multicolumn` functions as a row in your table. It takes 3 arguments: the number of columns it should span, a string dictating how it should be aligned, and a label that tells it what it should say:

```
\begin{center}
\begin{tabular}{ c|c }
    \hline
    \multicolumn{2}{|c|}{Undergraduate Voted}\\
    \hline
    Y $\downarrow$ & X = 1 \\
    \hline
    1 & $\alpha P(Y_g = 1)$ \\
    0 & $\beta (1-P(Y_g = 1))$\\
    \hline
    $P(X_g = 1)$ & $\alpha P(Y_g = 1) + \beta (1-P(Y_g = 1))$\\
\end{tabular}
\end{center}
```

Creates:

Undergraduate Voted	
Y $\downarrow$	X = 1
1	$\alpha P(Y_g = 1)$
0	$\beta(1 - P(Y_g = 1))$
$P(X_g = 1)$	$\alpha P(Y_g = 1) + \beta(1 - P(Y_g = 1))$

You can also easily do this for rows using the `\multirow` command in the `multirow` package. This command uses the same syntax as `\multicolumn` command. It's entered as a row in your table, and you tell LaTeX how many rows you want your label to span, how wide you'd like your label to be, and what your label is. For instance, I can format the table above to put the label for Y in its own multirow label:

```
\begin{center}
\begin{tabular}{ cc|c }
    \hline
    \multicolumn{3}{|c|}{Undergraduate Voted}\\
    \hline
    && X = 1 \\
    \hline
    \multirow{ 2}{*}{Y} & 1 & $\alpha P(Y_g = 1)$ \\
                        & 0 & $\beta (1-P(Y_g = 1))$ \\
    \hline
    $P(X_g = 1)$ & & $\alpha P(Y_g = 1) + \beta (1-P(Y_g = 1))$\\
\end{tabular}
\end{center}
```

Here, notice that I'm adding a third column to put the Y label in, and therefore stretching my span for the `\multicolumn` command. I'm also leaving blank column space for the label in the subsequent rows.

Undergraduate Voted		
		X = 1
Y	1	$\alpha P(Y_g = 1)$
	0	$\beta(1 - P(Y_g = 1))$
$P(X_g = 1)$		$\alpha P(Y_g = 1) + \beta(1 - P(Y_g = 1))$

Fancy Footnotes

One thing you may want to do is add footnotes to your table. The `threeparttable` package in LaTeX givesy ou a relatively simple way of doing that. `threeparttable` is an environment in which you can wrap your tabular environment. It also contains a `tablenotes` environment that lets you add footnotes to tables. Here's an example of how these work:

```
\begin{center}
```

```
\begin{threeparttable}
\begin{tabular}{ cc|c }
\hline
\multicolumn{3}{|c|}{Undergraduate Voted\tnote{1}}\\
\hline
 && X = 1 \\
\hline
\multirow{ 2}{*}{Y\tnote{2}} & 1 & $\alpha P(Y_g = 1)$ \\
   & 0 & $\beta (1-P(Y_g = 1))$ \\
\hline
$P(X_g = 1)$ & & $\alpha P(Y_g = 1) + \beta (1-P(Y_g = 1))$\\
\end{tabular}
\begin{tablenotes}
\item[1] We've given you fully conditional probabilities for
$P(X_g = 1 | Y_g = 1)$ and $P(Y_g = 1 | X_g = 1)$, so $\alpha$
and $\beta$ cannot be your joints by themselves
\item[2] Probabilities for $Y_g = 1$ are completely
symmetric to what you see for $Xg$
\end{tablenotes}
\end{threeparttable}
\end{center}
```

Undergraduate Voted[1]		
		X = 1
Y[2]	1	$\alpha P(Y_g = 1)$
	0	$\beta(1 - P(Y_g = 1))$
$P(X_g = 1)$		$\alpha P(Y_g = 1) + \beta(1 - P(Y_g = 1))$

[1] We've given you fully conditional probabilities for $P(X_g = 1|Y_g = 1)$ and $P(Y_g = 1|X_g = 1)$, so α and β cannot be your joints by themselves

[2] Probabilities for $Y_g = 1$ are completely symmetric to what you see for Xg

There are a few things worth noting in this code:

1) The footnotes are generated using the command \tnote^{number}

2) The tablenotes environment comes after the tabular environment is closed, but is still contained within your threeparttable environment.

3) The notes are generated using something very similar to an itemize environment. You can format it further as you might format text. Try wrapping all of the elements within the tablenotes environment in the statement {tiny}

Floats

Basic Float Placement

Floats are LaTeX environments that cannot be broken over multiple pages. Tables and figures are the most common types of floats, but you can create floats of other types in LaTeX if you need to. This section discusses how to control the locations of table and figure floats, like this one:

In order to have some control over the positions of your tables and figures, you'll need to wrap them in table or figure environments, as appropriate. These environments automatically look for "good" places for your tables and figures. They optimize for proximity to your original text and space availability to house the float. The code for the figure I've placed above wraps the \includegraphics statement in a figure environment:

```
\begin{figure}[h!]
   \begin{center}
   \includegraphics[scale=0.5]{floats.jpg}
   \end{center}
\end{figure}
```

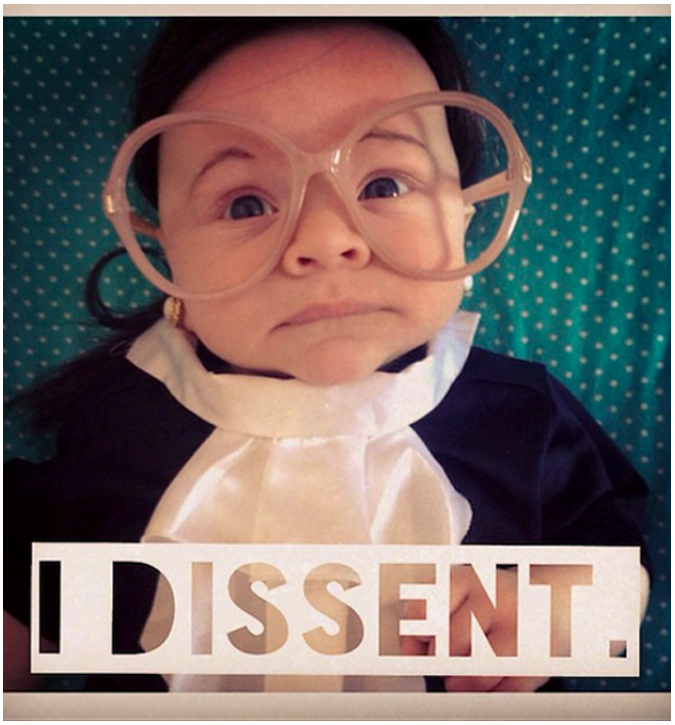
Notice the [h!] beside the \begin{figure} environment. This is an option that lets you select the position your figure appears in. This option can take on the following values:

Float Placement	Description
h	"here"; at approximately this point in the text if there is enough space
t	"top"; position float at the top of the page
b	"bottom"; position float at the bottom of the page
p	"page"; put this on a special page reserved just for floats
H	"HERE"; put this at exactly the place it occurs in the text
	note that this requires you to be running the package `float`
!	"dammit!"; override LaTeX's normal parameters

You can use combinations of these options. For instance [ht] tells LaTeX to place the float here if space is available, but if not put the float at the top of the page. The [h!] I'm using means "here, dammit!" I'm telling LaTeX to place the float here if at all possible and override all of LaTeX's normal conditions for optimizing my float location. These statements apply to `table` environments in the same way you see them used here. Here's a fun challenge: get into the LaTeX code for this template and move the figure you see above (we'll call him "Floatie") around on the page using these placement indicators).

Multiple Figure Environments

$\mathbb{R}$ makes it easy to print multiple graphics side by side or stacked on top of one another (we'll go over this again). But you may be importing images from other sources that you need to combine into a single image using LaTeX . That's also fairly simple. Suppose I have the following two images:


I DISSENT.

Okay yikes. That's not an efficient way to arrange my images. What I want is a way to arrange these images side by side. I obviously want to compare these, and it's much more efficient in terms of space to arrange them side by side. There are a number of ways to do this, but the `minipage` environment is one of the most intuitive. You're still placing all of your images in a `figure` environment, you're just wrapping that environment around several `minipage` environment. Here's how it works:

```
\begin{figure}[!h]
  \centering
  \begin{minipage}[t]{0.4\textwidth}
    \includegraphics[width=\textwidth]{blackwell.jpg}
    \caption{Matty B}
  \end{minipage}
  \hfill
  \begin{minipage}[t]{0.4\textwidth}
    \includegraphics[width=\textwidth]{ruthbabygins.jpg}
    \caption{Ruth Baby Ginsburg}
  \end{minipage}
\end{figure}
```

Figure 1: Matty B

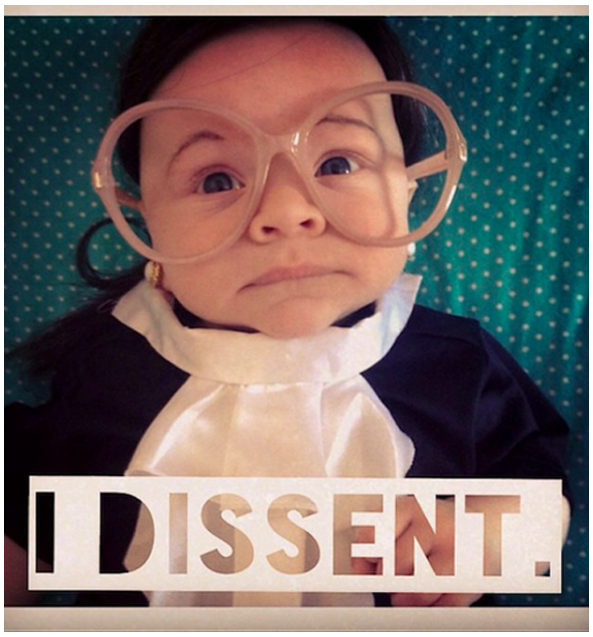


Figure 2: Ruth Baby Ginsburg

Much better. So what do all those options do? Notice that I'm creating `minipage` environments for each of my images within my `figure` environment. I'm positioning each `minipage` with its own positioning anchor; I've set both of these figures to the top. Minipage creates a horizontal band of space on your page to fill with figures. There is negative space (white space) surrounding each of these figures, so the anchor for each minipage tells LaTeX where in that horizontal space to put each image within your larger figure. The {`0.4\textwidth`} option you see in the code above tells LaTeX to set your minipage environments to 0.4 times the width of the text surrounding them. If you wanted to put three images across your page horizontally, you would reduce the proportion of the page each image took up (to around 0.32) and make sure to add an `\hfill` command after the end of each minipage environment to fill out the space around each image.

This environment also works with tables, you just need to place your `minipage` environments within a `table` environment and place your `tabular` environments inside of each `minipage`! Here's how it works:

```
\begin{table}[h!]
   \begin{minipage}[b]{0.45\textwidth}\centering
```

```
      \begin{tabular}{|c|c|}
          \hline
          &Vote Here!\\
          \hline
          Matty B &\\
          \hline
          Ruth Baby Ginsburg &\\
          \hline
      \end{tabular}
   \end{minipage}
   \hspace{0.5cm}
  \begin{minipage}[b]{0.45\textwidth}
   \centering
     \begin{tabular}{|c|c|}
         \hline
         &Vote There!\\
         \hline
         Matty B &\\
         \hline
         Ruth Baby Ginsburg &\\
         \hline
      \end{tabular}
   \end{minipage}
 \end{table}
```

Here you go:

	Vote Here!
Matty B	
Ruth Baby Ginsburg	

	Vote There!
Matty B	
Ruth Baby Ginsburg	

Labels and References

Giving your tables labels and captions is always a good idea, but LaTeX labels also have a specific purpose. The `hyperref` package lets you dynamically reference floats in your document. Regardless of the location of your floats, you can refer to them in text using their labels; if you use the correct syntax for this your in-text references will update even if the order of your tables changes. The first step is to add captions and labels to your float; note that these should be listed together:

```
\begin{figure}[h!]
\begin{center}
\includegraphics[scale=2.6]{survivor.jpg}
\caption{Survivor, Duh}
\label{fig:survivor}
\end{center}
\end{figure}
```

Figure 3: Survivor, Duh

Now I can refer to Figure 3 throughout my document using the syntax: `\ref{fig:survivor}`. Click on that reference, it will take you to the image! You can reference almost anything numbered in LaTeX . The general syntax for doing that follows these conventions:

- Label the float, section, subsection, table, item, or target object using the syntax `\label{ marker}`, where `marker` takes on:
  - ch:
  - sec:
  - subsec:
  - fig:
  - tab:
  - eq:
  - lst:
  - itm:

  - alg:
  - app:

- Refer to the object using \ref{type:label}, where type refers to one of the marker types listed directly above and label is the label you gave your object. You can also use \pageref{type:label} if you want to print the page number on which your object appears. For instance, Figure 3 appears on the previous page, page 27. See if you can move the float to another page and update the reference.

Printing Source Code in LaTeX

This section introduces two environments that will make it easier for you to copy and paste your ℝ code into LaTeX without worrying about compilation errors. There are a couple of ways to get LaTeX to print source code as it appears (including LaTeX's own source code!).

The Verbatim Environment

The verbatim environment prints everything contained within it exactly as it appears, including reserved characters in LaTeX . I've been using it to display LaTeX code throughout this document. Note that the verbatim environment doesn't read tabs; anything you want indented has to be moved with spaces and hard returns. This is how it works:

```
Here are a bunch of special characters that would break LaTeX if they
weren't in a verbatim environment:
\ ^ & $ _ { }
Also here is some R code:
object <- rpois(1000, lambda = 9)
# and some comments
```

The text displayed above is wrapped in \begin{verbatim} and \end{verbatim}

Listings

If you scroll up to the preamble of this document, you'll see that I'm running the package `listings` and have a bunch of stylistic preferences defined in an option called `lstset`. There is some discussion of `listings` in the packages section, but this package lets you print source code from a variety of different programs. `listings` is more faithful to the original formatting of your source code, and more customizable for appearance. It also lets you use characters that are reserved in LaTeX without escaping them. To use this environment, you just need to wrap your code in \begin{lstlisting} and \end{lstlisting} tags! `listings` *does* read tabs as tabs. You can copy and paste your code right into it. The result looks like this:

```
# Basic plot:
    # Print
      pdf("pricehist.pdf")
      hist(data$PRICE, main = "Histogram of Home Price Data",
      xlab = "Price", col = "dodgerblue2", border = "white", breaks = seq
         (500, 2500, 150))
      dev.off()

  # ggplot:
    require(ggplot2) # don't have this? use install.packages("ggplot2")
       before you require
    p <- ggplot(data = data, aes(x = PRICE))
    q <- p + geom_histogram(binwidth = 100, colour = "white", fill = "
       dodgerblue2") +
      xlab("Home Price") + ylab("Frequency") + ggtitle("Histogram of Home
         Price Data")
    q
```

Relative References

By default, LaTeX looks for graphics and tables you input within the same directory that houses your .tex file. But you can put your graphics and tables wherever you want! You just have to adjust the pathnames you use in \includegraphics, \input and other import commands to tell LaTeX where to search for things. That's called relative referencing. Here's

how it works:

I have an "Other Graphics" folder in this demo from which I'd like to import an image. Let's say I want to import the `ghits.pdf` file in that folder. I can tell LaTeX to get out of my working directory using standard UNIX notation: one dot . refers to my current working directory; two dots .. tells LaTeX to look in the previous working directory, and forward slashes / split the directory path like they normally would. So:

- ./ refers to my current directory
- ../ refers to the previous working directory
- ../../ refers to a directory that is two steps above my current working directory

And I would import my graphic in another folder using this syntax:

```
\begin{figure}[h!]
\begin{center}
\includegraphics[scale=0.7]{{"../Other Graphics/ghits"}.pdf}
\caption{Self Googling}
\label{fig:ghits}
\end{center}
\end{figure}
```

I'm wrapping the filename in quotes and extra braces before the file extension above to prevent LaTeX from printing the filepath below the image. Your code will compile if you don't do this, but it will show the filename and extension below the image every time.

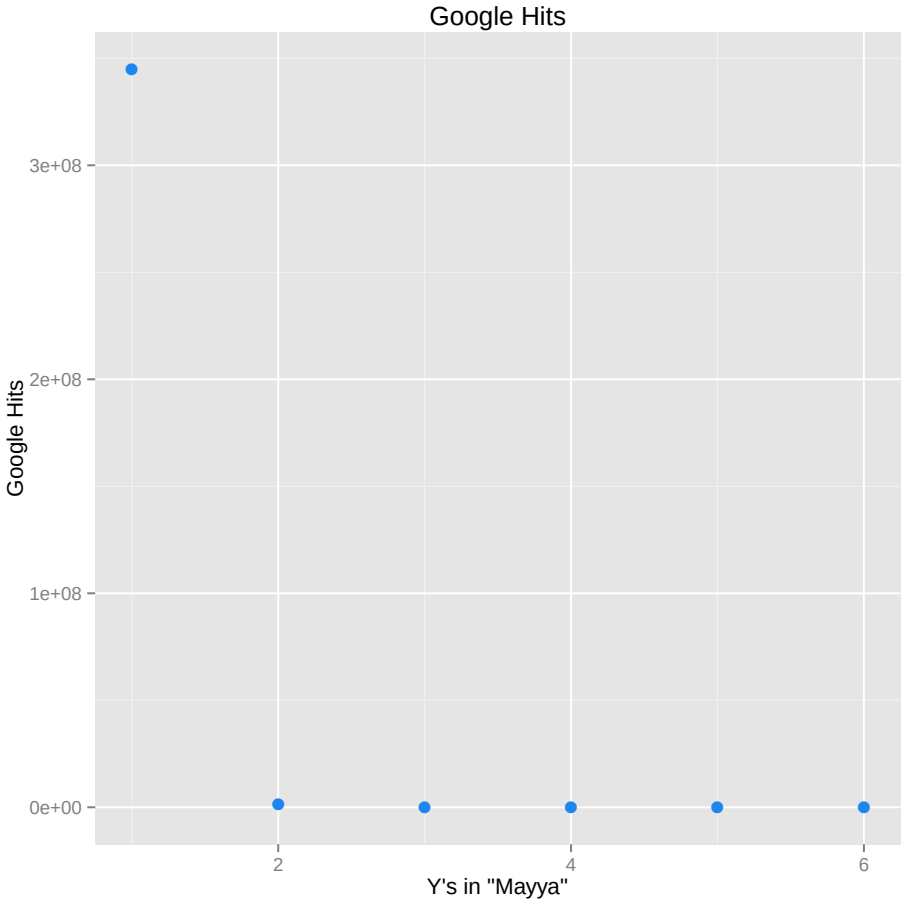


Figure 4: Self Googling

Stargazing

Let's import some tables from ℝtext directly using the `\input` command.

Default Stargazer Table for a Regression Object

Table 3:

	Dependent variable:
	y
x1	13.982***
	(0.017)
x2	2.999***
	(0.001)
Constant	6.177***
	(0.205)
Observations	10,000
R^2	1.000
Adjusted R^2	1.000
Residual Std. Error	4.954 (df = 9997)
F Statistic	12,058,683.000*** (df = 2; 9997)
Note:	*p<0.1; **p<0.05; ***p<0.01

Labeled Variables

Table 4: The Most Wonderful Time of the Year

	Time to Sit Back and Unwind?
	Swimming Pool Deaths
Ice Cream Bars Sold	13.982***
	(0.017)
Temperature	2.999***
	(0.001)
Constant	6.177***
	(0.205)
Observations	10,000
R^2	1.000
Adjusted R^2	1.000
Residual Std. Error	4.954 (df = 9997)
F Statistic	12,058,683.000*** (df = 2; 9997)
Note:	*p<0.1; **p<0.05; ***p<0.01

Modifying Reported Statistics

Table 5: The Most Wonderful Time of the Year

	Time to Sit Back and Unwind?
	Swimming Pool Deaths
Ice Cream Bars Sold	13.982
	(0.017)
	t = 841.609
	p = 0.000***
Temperature	2.999
	(0.001)
	t = 4,833.702
	p = 0.000***
Constant	6.177
	(0.205)
	t = 30.123
	p = 0.000***
Observations	10,000
R^2	1.000
Adjusted R^2	1.000
Residual Std. Error	4.954 (df = 9997)
F Statistic	12,058,683.000*** (df = 2; 9997)
Note:	*p<0.1; **p<0.05; ***p<0.01

Table 6: The Most Wonderful Time of the Year

	Swimming Pool Deaths
Ice Cream Bars Sold	13.982***
	(0.017)
Temperature	2.999***
	(0.001)
Constant	6.177***
	(0.205)
N	10,000
R^2	1.000
Adjusted R^2	1.000
Residual Std. Error	4.954 (df = 9997)
F Statistic	12,058,683.000*** (df = 2; 9997)

*p < .1; **p < .05; ***p < .01

Modifying Spacing

Table 7: The Most Wonderful Time of the Year

	Time to Sit Back and Unwind?
	Swimming Pool Deaths
Ice Cream Bars Sold	13.982 (0.017)
	t = 841.609
	p = 0.000***
Temperature	2.999 (0.001)
	t = 4,833.702
	p = 0.000***
Constant	6.177 (0.205)
	t = 30.123
	p = 0.000***
Observations	10,000
R^2	1.000
Adjusted R^2	1.000
Residual Std. Error	4.954 (df = 9997)
F Statistic	12,058,683.000*** (df = 2; 9997)
Note:	*p<0.1; **p<0.05; ***p<0.01

Modifying Floats

Table 8: The Most Wonderful Time of the Year

	Time to Sit Back and Unwind?
	Swimming Pool Deaths
Ice Cream Bars Sold	13.982 (0.017)
	t = 841.609
	p = 0.000***
Temperature	2.999 (0.001)
	t = 4,833.702
	p = 0.000***
Constant	6.177 (0.205)
	t = 30.123
	p = 0.000***
Observations	10,000
R^2	1.000
Adjusted R^2	1.000
Residual Std. Error	4.954 (df = 9997)
F Statistic	12,058,683.000*** (df = 2; 9997)
Note:	*p<0.1; **p<0.05; ***p<0.01