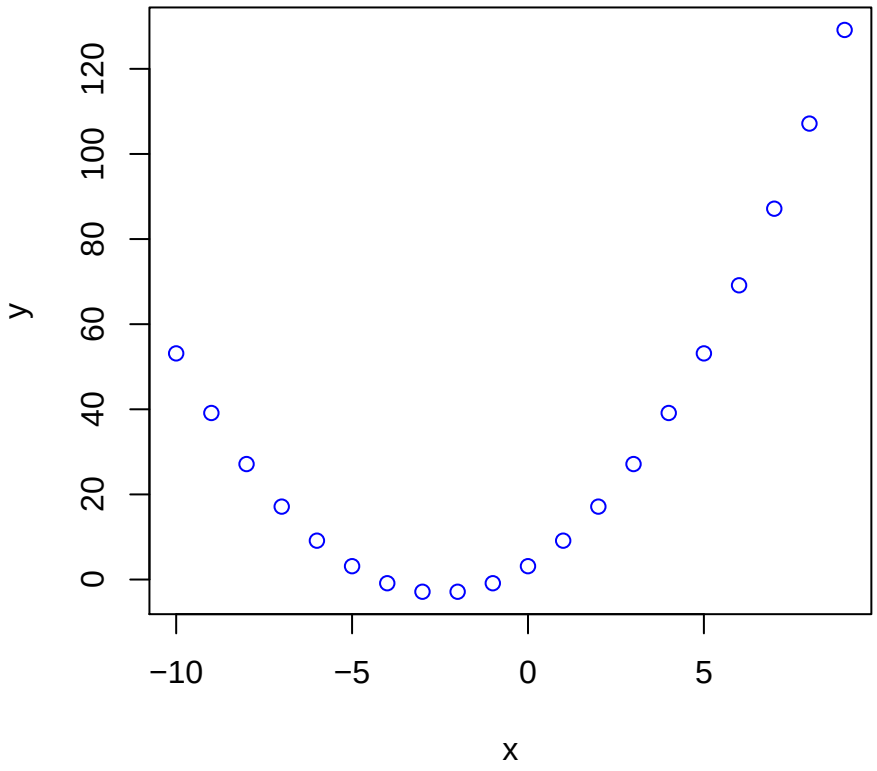
Makefiles

Originally designed for compiling large software projects, Makefiles can automate the dependencies between data analysis steps, enabling **reproducible research**.[1] Taking a little time early on to precisely record and understand your steps in a Makefile will pay big dividends when it comes time to reproduce that analysis.

To create this pdf, navigate to the directory containing the Makefile and type `make`. Note that you will need LaTeX, R, and Python installed for this example. The Makefile contains a list of dependencies and commands that act as a recipe for building the final product, in this case a pdf. Later, if you change any step in the analysis you can issue the `make` command again and the make program will run the appropriate commands to refresh the results.

Here we have the result of our labors, a nice plot from R. PDF's are vector graphics, which means that they'll look good no matter how much you zoom. Use them for professional results. Mathematical expressions are always beautiful in LaTeX. $x^2 + 5x + \pi$



[1]For more on reproducible research, check out the chapter on Open Source Scientific Practice by K. Jarrod Millman and Fernando Perez available at `https://osf.io/h9gsd/`.

Makefiles consist of rules that follow this pattern:

```
target: dependencies
[tab] system command
```

Note that it **must** have a tab character, spaces will not work. You may need to change your text editor settings. This is one of the only times that actual tab characters are a good idea when programming. The Makefile below generates this pdf.

Makefile

```
paper.pdf : paper.tex figure.pdf make_graph.pdf Makefile
        pdflatex paper.tex

# Running vanilla means that R won't save or use .RData files
figure.pdf : data.csv plot.R
        R CMD BATCH --vanilla plot.R

data.csv : get_data.py
        python get_data.py

make_graph.pdf : make_graph.py Makefile
        python make_graph.py > make_graph.dot
        dot -Tpdf make_graph.dot -o make_graph.pdf

clean :
        rm *.Rout *.aux *.log
```

The first line describes our final target, `paper.pdf`. It says that `paper.pdf` depends on the TEX source `paper.tex`, `figure.pdf`, and `Makefile`. If any of these three files change then the corresponding command `pdflatex paper.tex` will run, which makes a new `paper.pdf`. It's a little strange that it depends on `Makefile`, but the reason is that LATEX actually pulls in the source code of `Makefile` and renders it in the pdf!

The line with `figure.pdf` as the target has dependencies `data.csv` and `plot.R`. If the data or the R script changes then `make` needs to create a new figure.

`data.csv` is a target with dependency `get_data.py`. If the Python script that produces the data changes and we run `make` again then the command `python get_data.py` will run, updating `data.csv`. This will cause `figure.pdf` to be plotted again, which will in turn cause the final pdf to be recreated.

`make` is great because it's lazy. Good programmers always look out for a chance to be lazy. That means that if you only change `paper.tex`, then `make` won't plot the figure or do

anything with the python script.

The final target is `clean`. Issuing the command `make clean` runs the line beginning with `rm`, removing the byproducts of R and LaTeX.

Below is a visualization of the Makefile. It was automatically created using the dot language, aka Graphviz.