- There are **many** ways to link Python and R
- *How* is often very easy, and the best way to learn about is by documentation+doing (as usual).
- This presentation is about *when* to do *what*, and *why*.

- The canonical use cases
  - Exploratory/one-off data analysis → R/Rstudio
  - Automated algorithm application as part of another system → Python
  - More complex multistage data analysis → sequence of Python and R scripts (sometimes ending in an Sweave document)

- But
  - You can use Python to explore data (e.g. with SciPy)
  - You can run R code as shell scripts (`#!/usr/bin/Rscript`)
- You need to mix less often than you think: esoteric maths that call for R and harsh engineering that demands Python (or for fun).

Scenario 1: Python codebase, you need to add non-trivial stats

- If it's real-time or interactive → rpy/rpy2
  - `import rpy2.robjects as robjects`
  - `Robjects.r('2+2')`
  - `And so on and so forth`
- Otherwise, consider calling an R script
  - Separation of concerns/clear interfaces
  - Saves a lot of debugging time

Scenario 2: existing R code needs access to Python features

- `System() # low granularity`
- `read.table.url() # data access`
- `httr package → POST, GET, etc`
  - HTTP over TCP over IP is history's most successful integration technology. Scalability, distribution, etc, come almost for free.
- If it can't be done like this, most likely you're talking C anyway.

Scenario 3: server-side

- Databases for shared representation
  - SQL, MongoDB, ...
  - Annotation, enhancement, prediction
- Hadoop jobs in R (streaming or more sophisticated Revolution tech)
- Shiny (not really Python-related, but sometimes makes it unnecessary)

Takeaways

- Integrating programs is an OS concern – more often than not, files and networking protocols are the right choice.
- Mixed code is easy in the simple cases, and probably overkill in the complex ones.
- In industrial practice, the infrastructure provides the integration bus.