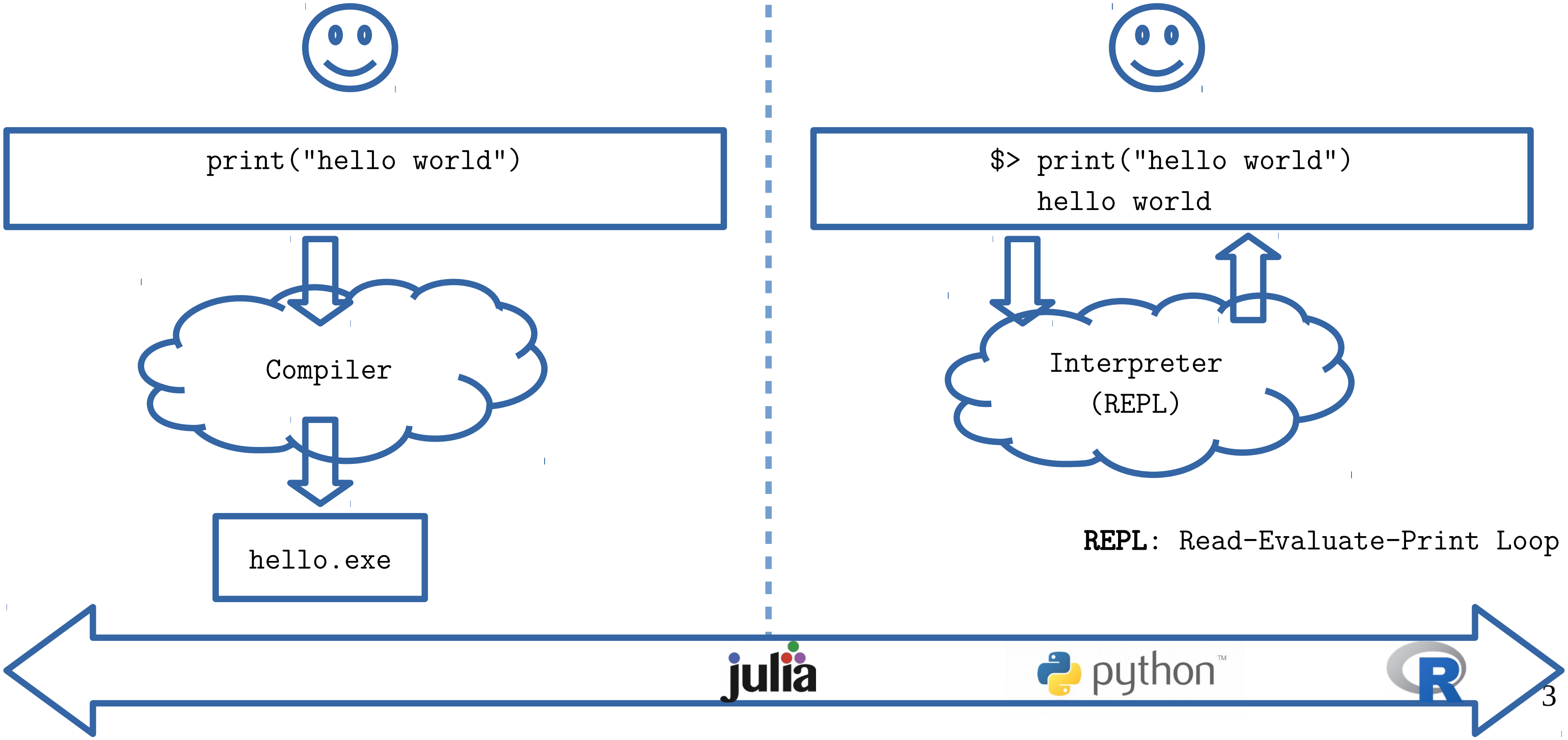
R, Python, Julia: do you know them all?

Mark van der Loo
Statistics Netherlands

EMOS webinar, 24 March 2020

Contents

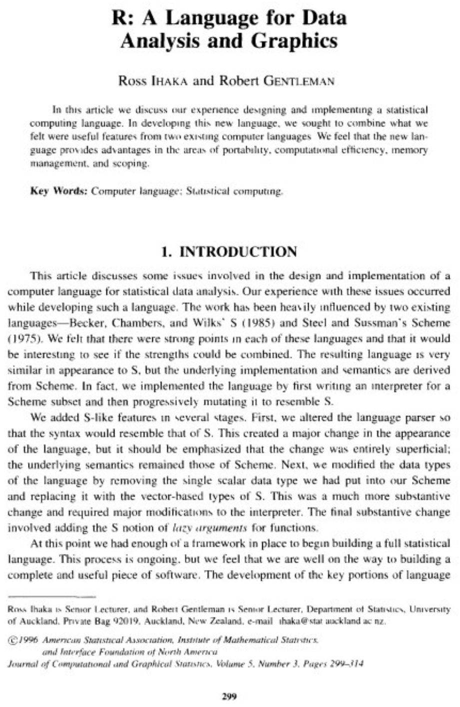
Compiled vs Interpreted



R

“R is a free software environment for ***statistical computing and graphics****.”*

- Started as free implementation of S (est. 1976, Bell Labs) at Univ. of Auckland in 1992.
- V1.0: 29-02-2000, V4.0: 2020.
- GNU GPL (part of GNU)



R: A Language for Data Analysis and Graphics

Ross IHAKA and Robert GENTLEMAN

In this article we discuss our experience designing and implementing a statistical computing language. In developing this new language, we sought to combine what we felt were useful features from two existing computer languages. We feel that the new language provides advantages in the areas of portability, computational efficiency, memory management, and scoping.

Key Words: Computer language; Statistical computing.

1. INTRODUCTION

This article discusses some issues involved in the design and implementation of a computer language for statistical data analysis. Our experience with these issues occurred while developing such a language. The work has been heavily influenced by two existing languages—Becker, Chambers, and Wilks' S (1985) and Steel and Sussman's Scheme (1975). We felt that there were strong points in each of these languages and that it would be interesting to see if the strengths could be combined. The resulting language is very similar in appearance to S, but the underlying implementation and semantics are derived from Scheme. In fact, we implemented the language by first writing an interpreter for a Scheme subset and then progressively mutating it to resemble S.

We added S-like features in several stages. First, we altered the language parser so that the syntax would resemble that of S. This created a major change in the appearance of the language, but it should be emphasized that the change was entirely superficial; the underlying semantics remained those of Scheme. Next, we modified the data types of the language by removing the single scalar data type we had put into our Scheme and replacing it with the vector-based types of S. This was a much more substantive change and required major modifications to the interpreter. The final substantive change involved adding the S notion of *lazy arguments* for functions.

At this point we had enough of a framework in place to begin building a full statistical language. This process is ongoing, but we feel that we are well on the way to building a complete and useful piece of software. The development of the key portions of language

Ross Ihaka is Senior Lecturer, and Robert Gentleman is Senior Lecturer, Department of Statistics, University of Auckland, Private Bag 92019, Auckland, New Zealand, e-mail ihaka@stat.auckland.ac.nz.

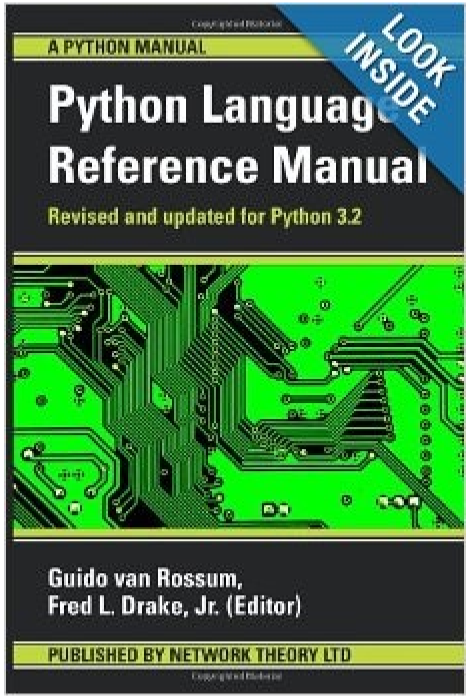
Journal of Computational and Graphical Statistics, Volume 5, Number 3, Pages 299–314

R. Ihaka and Gentleman, R (1996). JCGS **5(3)** 299-314

Python

“*Python is an interpreted, object-oriented, **high-level programming language** with dynamic semantics.*”

- Started at CWI (Amsterdam) in 1989
- V1.0 194, V2.0 1998, V3.0 2008
- PSF Licence (GPL compatible)

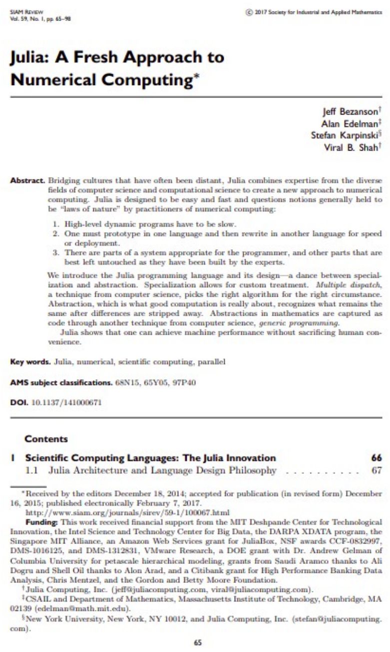


G. Van Rossum and Drake, F.L. *Python Language Reference Manual*

Julia

``Julia combines expertise from the diverse fields of computer science and computational science to create a new approach to ***numerical computing****."*

- Developed at MIT since 2009
- V1.0 2018
- MIT licence


SIAM Review
Vol. 59, No. 1, pp. 65–98

Julia: A Fresh Approach to Numerical Computing*

Jeff Bezanson[†]
Alan Edelman[‡]
Stefan Karpinski[§]
Viral B. Shah[†]

Abstract. Bridging cultures that have often been distant, Julia combines expertise from the diverse fields of computer science and computational science to create a new approach to numerical computing. Julia is designed to be easy and fast and questions notions generally held to be "laws of nature" by practitioners of numerical computing:

1. High-level dynamic programs have to be slow.
2. One must prototype in one language and then rewrite in another language for speed or deployment.
3. There are parts of a system appropriate for the programmer, and other parts that are best left untouched as they have been built by the experts.

We introduce the Julia programming language and its design—a dance between specialization and abstraction. Specialization allows for custom treatment. *Multiple dispatch*, a technique from computer science, picks the right algorithm for the right circumstance. Abstraction, which is what good computation is really about, recognizes what remains the same after differences are stripped away. Abstractions in mathematics are captured as code through another technique from computer science, *generic programming*.

Julia shows that one can achieve machine performance without sacrificing human convenience.

Key words. Julia, numerical, scientific computing, parallel

AMS subject classifications. 68N15, 65Y05, 97P40

DOI. 10.1137/141000671

Contents

*Received by the editors December 18, 2014; accepted for publication (in revised form) December 16, 2015; published electronically February 7, 2017.
http://www.siam.org/journals/sirev/59-1/100067.html

Funding: This work received financial support from the MIT Deshpande Center for Technological Innovation, the Intel Science and Technology Center for Big Data, the DARPA XDATA program, the Singapore MIT Alliance, an Amazon Web Services grant for JuliaBox, NSF awards CCF-0832997, DMS-1016125, and DMS-1312831, VMware Research, a DOE grant with Dr. Andrew Gelman of Columbia University for petascale hierarchical modeling, grants from Saudi Aramco thanks to Ali Dogru and Shell Oil thanks to Alon Arad, and a Citibank grant for High Performance Banking Data Analysis, Chris Mentzel, and the Gordon and Betty Moore Foundation.

[†]Julia Computing, Inc. (jeff@juliacomputing.com, viral@juliacomputing.com).
[‡]CSAIL and Department of Mathematics, Massachusetts Institute of Technology, Cambridge, MA 02139 (edelman@math.mit.edu).
[§]New York University, New York, NY 10012, and Julia Computing, Inc. (stefan@juliacomputing.com).

Bezanson *et al.* (2017)
Siam Review **59** 65-98

Example: iris data

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.1	3.5	1.4	0.2	setosa
2	4.9	3.0	1.4	0.2	setosa
3	4.7	3.2	1.3	0.2	setosa
4	4.6	3.1	1.5	0.2	setosa
5	5.0	3.6	1.4	0.2	setosa
6	5.4	3.9	1.7	0.4	setosa

Showing 6 / 150 records

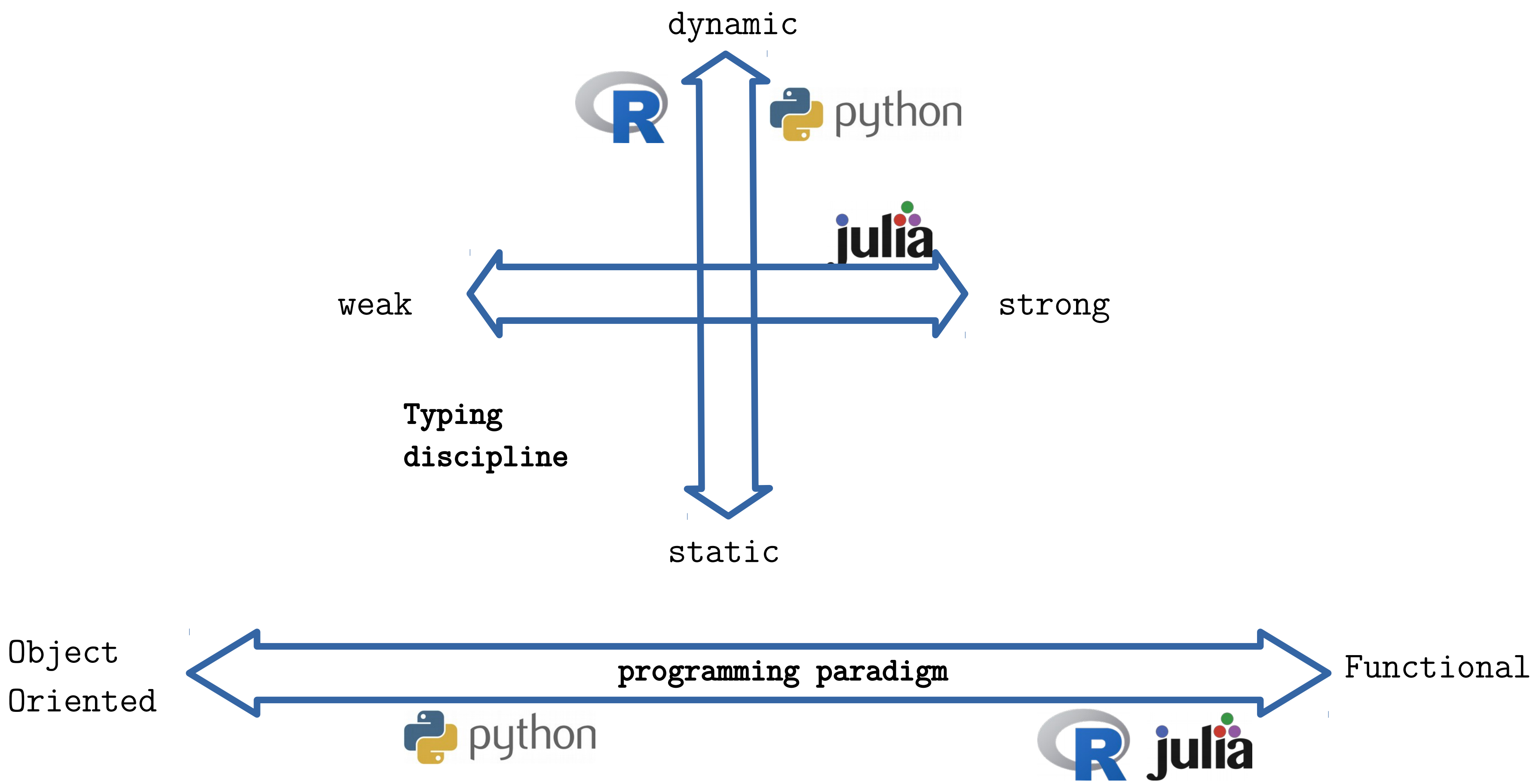
Example: grouped aggregation

```
> iris <- read.csv("iris.csv")
> aggregate(iris["Sepal_Length"], by=iris["Species"], mean)
     Species Sepal_Length
1     setosa        5.006
2 versicolor        5.936
3  virginica        6.588
```

```
>>> import pandas as pd
>>> iris = pd.read_csv("iris.csv")
>>> iris.groupby("Species")[["Sepal_Length"]].agg("mean")
```

```
julia> using CSV, DataFrames, GLM
julia> iris = DataFrame(CSV.File("iris.csv"));
julia> by(iris, :Species, :Sepal_Width => mean)
```

Rough comparison



For data analyses

Feature	R	python	julia
Missing value support	native	library	library
Data frame	native	library	library
Interactive graphics	native	library	library
Statistics	native	library	library
Machine Learning	library	library	library

$[001]> Questions_

Packages

	Publishing	#libraries
R	CRAN	15k
python	PyPi	224k
julia	General Registry	3.3k

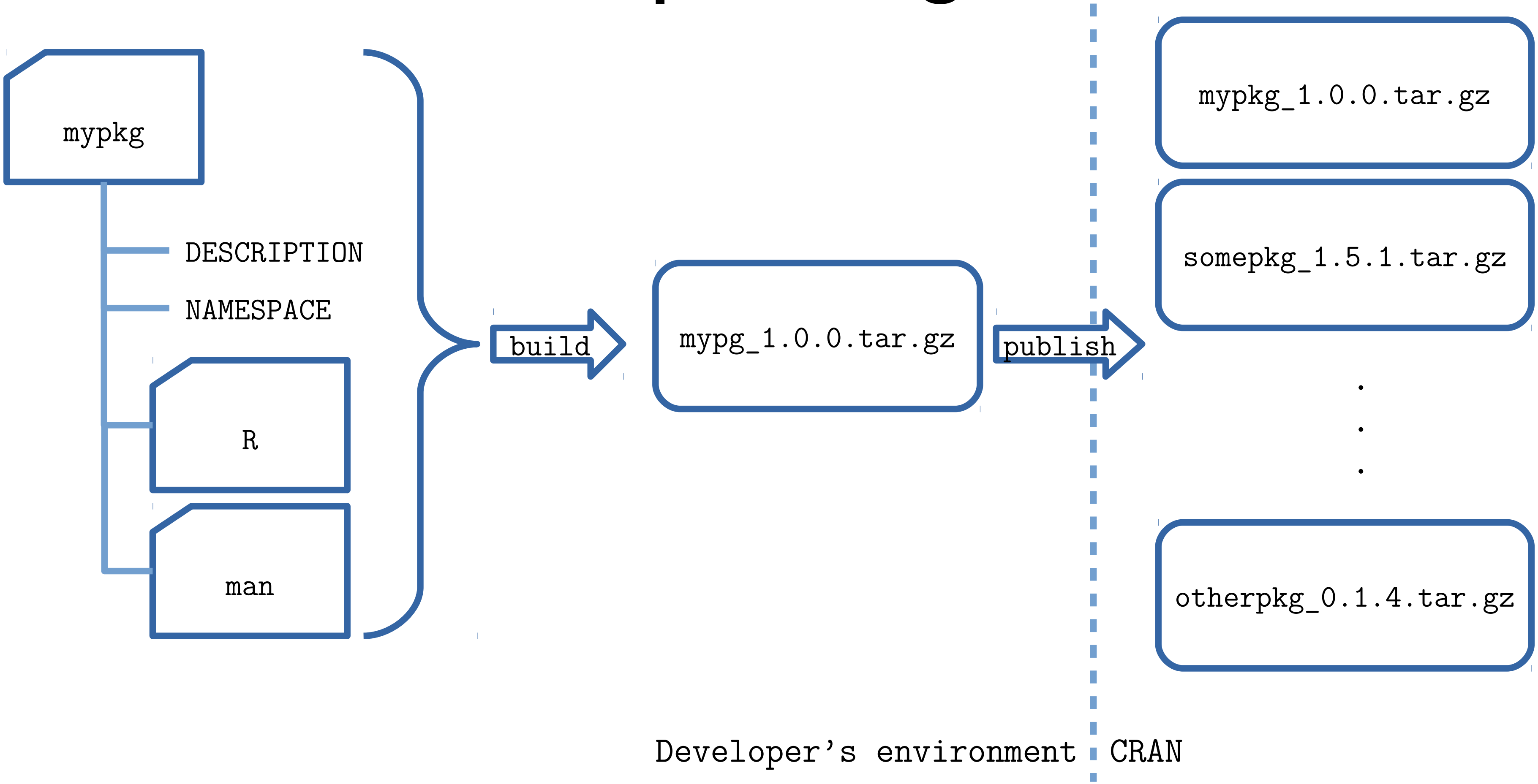
Installing and loading a package

```
> install.packages("mypkg")
> library(mypkg)
```

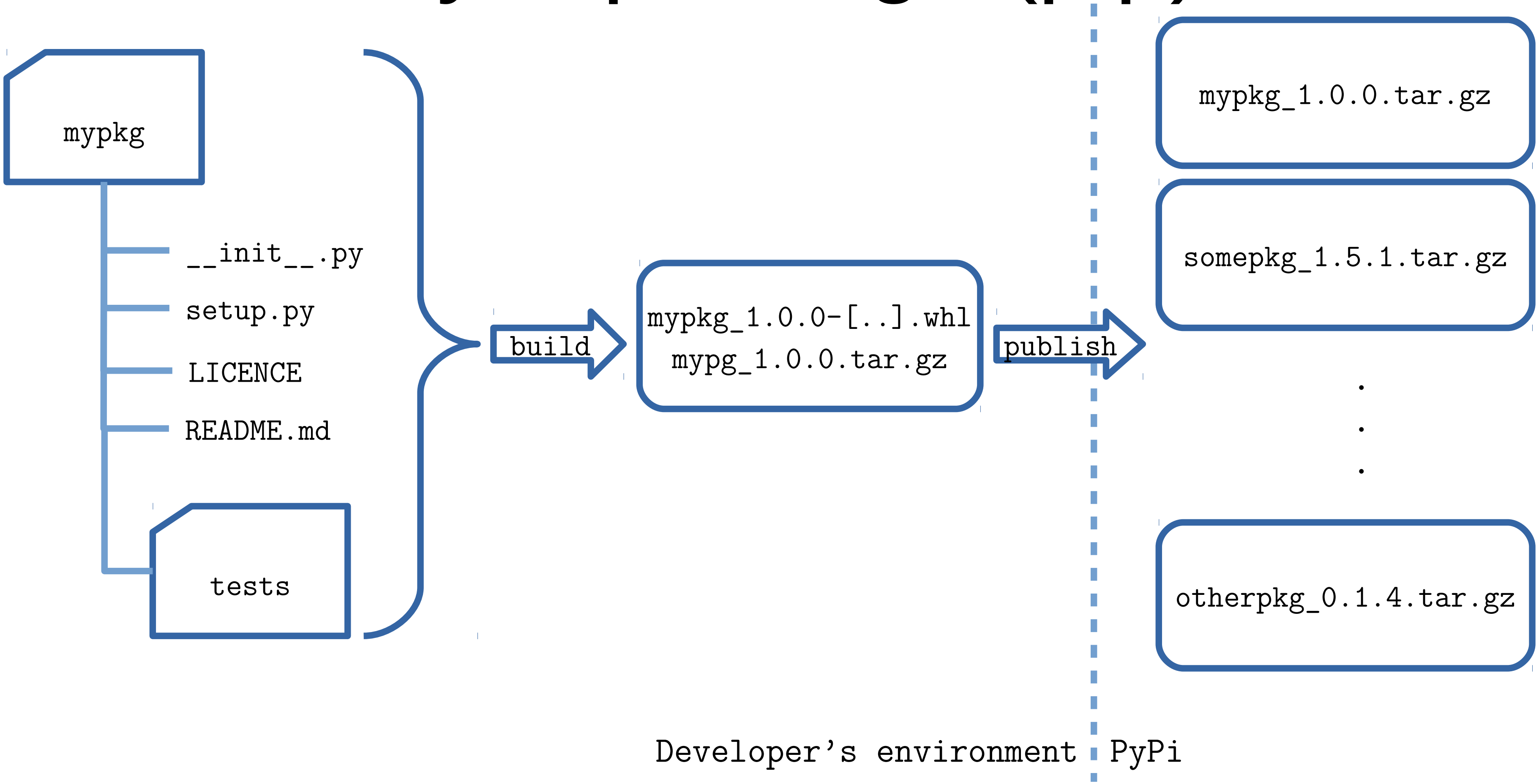
```
$> pip install mypkg
$> python3
>>> import mypkg
```

```
julia> using Pkg
julia> Pkg.add("Mypkg")
julia> using Mypkg
```

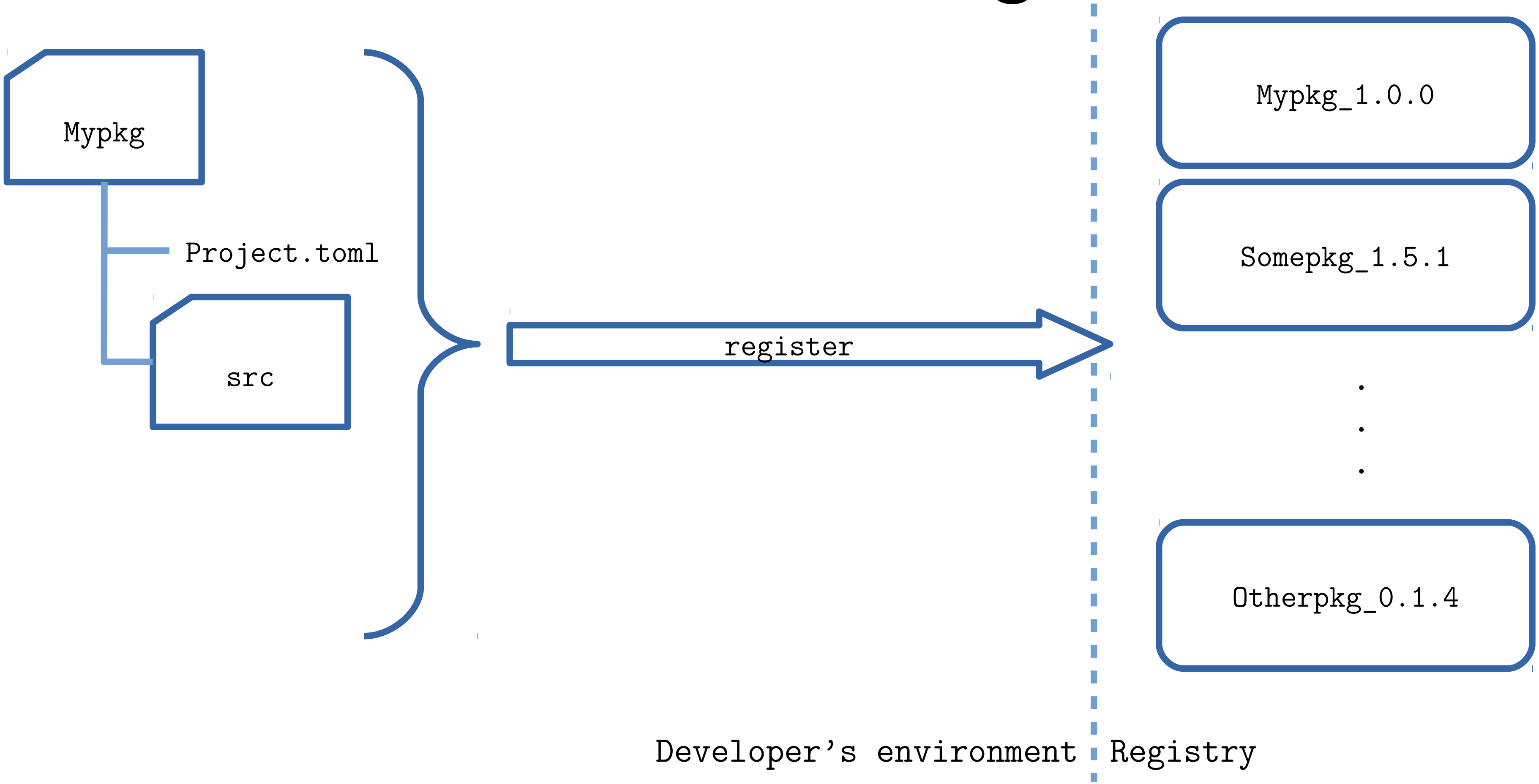
R package


mypkg
DESCRIPTION
NAMESPACE
R
man
build
mypg_1.0.0.tar.gz
publish
mypkg_1.0.0.tar.gz
somepkg_1.5.1.tar.gz
otherpkg_0.1.4.tar.gz
Developer's environment
CRAN

PyPi package (pip)



Julia Package



Package publishing systems

		Registry	Hosting	Checking
R	CRAN	yes	yes	yes
python	PyPi	yes	yes	no*
julia	Developer's Git repo.	yes	no	no*

* Except some very basic checks on metadata

CRAN checks

- **On 1st publication**

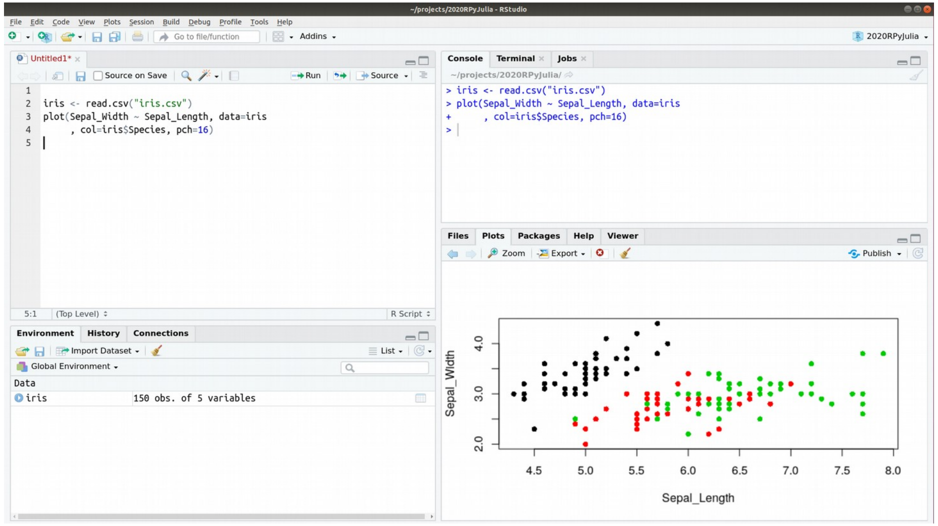
  Human checks on relevance, intellectual property, description + all automated checks. Also: no anonymous packages allowed.

- **On updates**

  Automated checks on:
  - Package integrity
  - R code validity
  - Documentation/code consistency
  - Examples
  - Unit tests (if any)
  - Platform independence
  - Cross-package integrity(!): dependencies and reverse dependencies

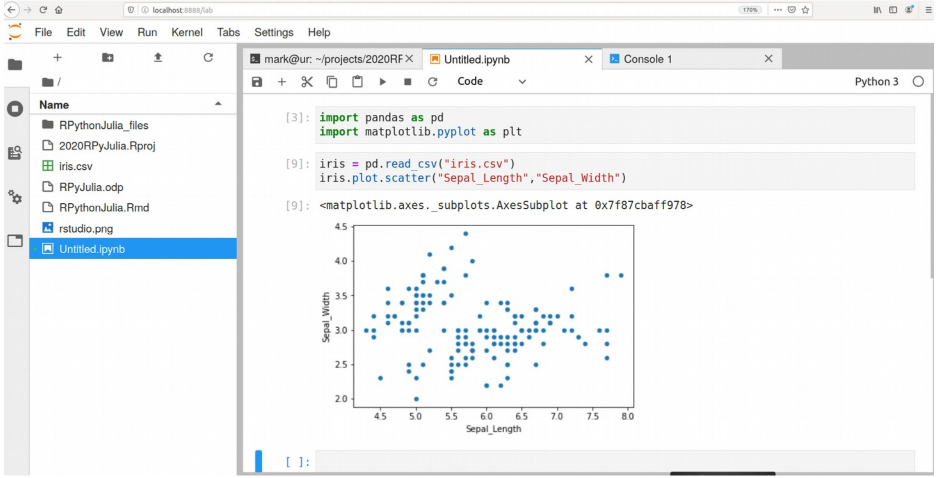
$[002]> Questions_

Working with data: RStudio



- Data science workbench
- Integrated R console, plots, help, file browser, environment browser, git support,job scheduling, connection browser, ...
- Native support for R, SQL, JS, HTML, markdown, yaml, tex
- Python console

Working with data: Jupyter lab



- Data science notebook
- Supports Julia, Python, R, markdown
- Runs in browser
- Exports to many different report formats

Data analyses: linear models

```
> iris <- read.csv("iris.csv")
> model <- lm(Sepal_Length ~ Sepal_Width, data=iris)
```

```
>>> import pandas as pd
>>> from sklearn.linear_model import LinearRegression
>>> iris = pd.read_csv("iris.csv")
>>> model = LinearRegression().fit(
        iris.Sepal_Length.values.reshape(-1,1)
      , iris.Sepal_Width.values.reshape(-1,1))
```

```
julia> using DataFrames, CSV, GLM
julia> iris = DataFrame(CSV.File("iris.csv"))
julia> model = lm(@formula(Sepal_Length ~ Sepal_Width), iris)
```

Models (native)

	R	**Python**	**Julia**
(G)LM	yes	yes	yes
Regularized regression	yes	yes	yes
CART	yes	yes	yes
Random Forest, (X)GB	yes	yes	yes
Deep Learning	no	**yes**	sortof
SVM	yes	**yes**	**yes**
Clustering	yes	yes	yes

Task-based packages

	R	Python	julia
Complex sampling and weighting	several	few	no
Disclosure control	several	no	no
Small Area estimation	several	no	no
Imputation	many (> 100)	few	few
Time series/ seasonal adjustment / arima	many (>100)	few	few

Commonalities

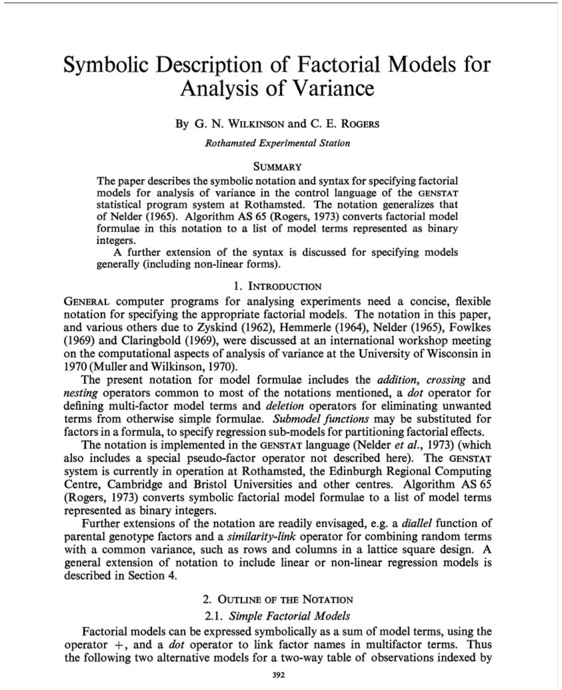
- Data frames
  - R: native
  - Python: **pandas**
  - Julia: **DataFrames**
- Formula-data interface
  - Python: **pandas** + **patsy** + **statsmodels/sklearn**
  - Julia: **DataFrames** + **GLM**
- Grammar of Graphics
  - R: **ggplot2**
  - Python: **plotnine**
  - Julia: **Gadfly**
- Data manipulation
  - R: native + **dplyr**, **data.table**
  - Python: **Pandas**
  - Julia: **DataFrames**

Formula notation for models

- S/R: Native
- Python: **patsy**
- Julia: **DataFrames**
- Example:

$$Y = b_0 + b_1 X_1 + b_2 X_2 + b_{12} X_1 X_2$$

```
Y ~ X1 + X1 + X1*X2
```



Symbolic Description of Factorial Models for Analysis of Variance

By G. N. WILKINSON and C. E. ROGERS

Rothamsted Experimental Station

SUMMARY

The paper describes the symbolic notation and syntax for specifying factorial models for analysis of variance in the control language of the GENSTAT statistical program system at Rothamsted. The notation generalizes that of Nelder (1965). Algorithm AS 65 (Rogers, 1973) converts factorial model formulae in this notation to a list of model terms represented as binary integers.

A further extension of the syntax is discussed for specifying models generally (including non-linear forms).

1. INTRODUCTION

GENERAL computer programs for analysing experiments need a concise, flexible notation for specifying the appropriate factorial models. The notation in this paper, and various others due to Zyskind (1962), Hemmerle (1964), Nelder (1965), Fowlkes (1969) and Claringbold (1969), were discussed at an international workshop meeting on the computational aspects of analysis of variance at the University of Wisconsin in 1970 (Muller and Wilkinson, 1970).

The present notation for model formulae includes the *addition*, *crossing* and *nesting* operators common to most of the notations mentioned, a *dot* operator for defining multi-factor model terms and *deletion* operators for eliminating unwanted terms from otherwise simple formulae. *Submodel functions* may be substituted for factors in a formula, to specify regression sub-models for partitioning factorial effects.

The notation is implemented in the GENSTAT language (Nelder *et al.*, 1973) (which also includes a special pseudo-factor operator not described here). The GENSTAT system is currently in operation at Rothamsted, the Edinburgh Regional Computing Centre, Cambridge and Bristol Universities and other centres. Algorithm AS 65 (Rogers, 1973) converts symbolic factorial model formulae to a list of model terms represented as binary integers.

Further extensions of the notation are readily envisaged, e.g. a *diallel* function of parental genotype factors and a *similarity-link* operator for combining random terms with a common variance, such as rows and columns in a lattice square design. A general extension of notation to include linear or non-linear regression models is described in Section 4.

2. OUTLINE OF THE NOTATION

2.1. *Simple Factorial Models*

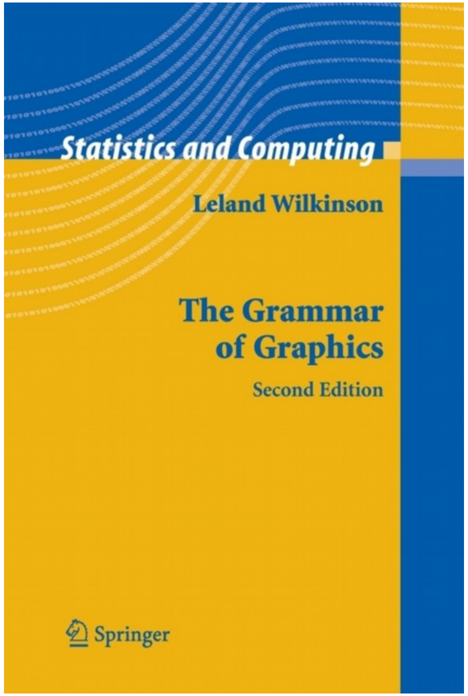
Factorial models can be expressed symbolically as a sum of model terms, using the operator +, and a *dot* operator to link factor names in multifactor terms. Thus the following two alternative models for a two-way table of observations indexed by

392

GN Wilkinson and CE Rogers (1973) Applied Statistics **22** 392-399

Grammar of Graphics

- R: **ggplot2**
- Python: **plotnine**
- Julia **Gadfly**
- Main Idea:
  - Map data attributes to aesthetic and geometric properties of a graphic.

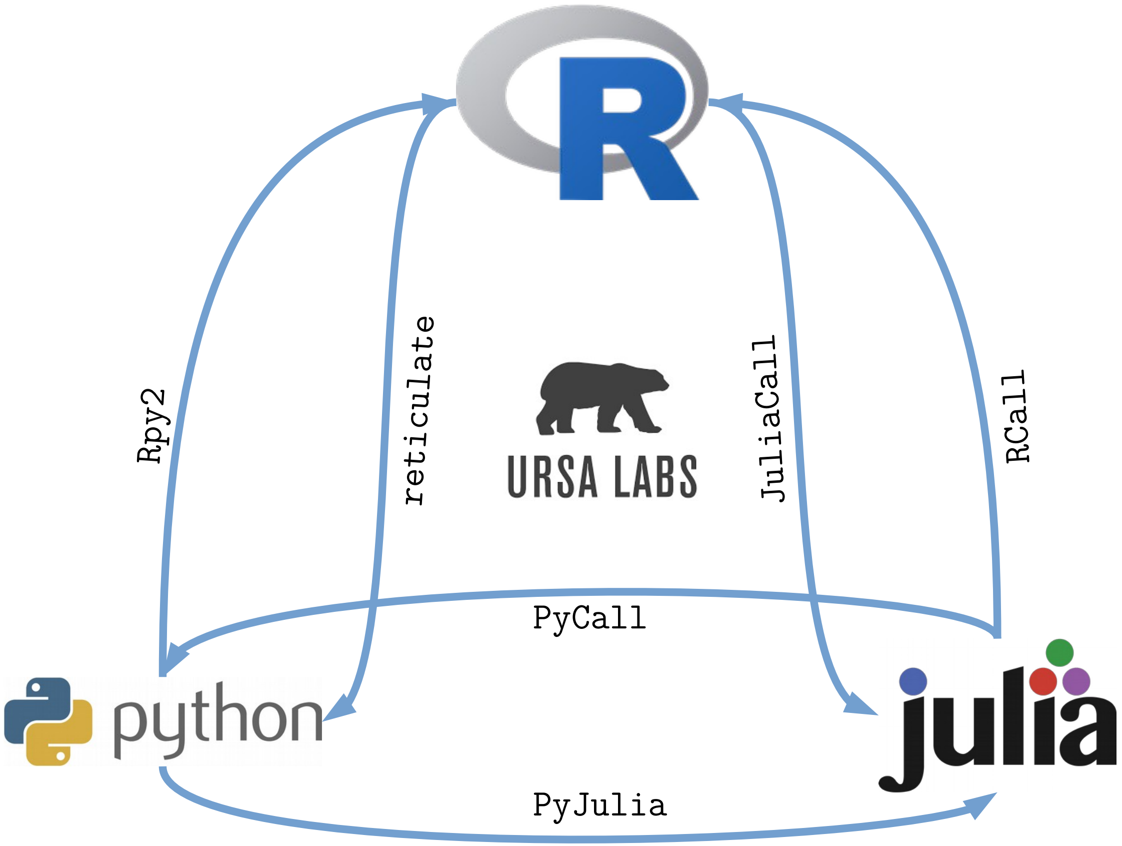


L Wilkinson (1999) Springer.

Some conclusions

- There are technical differences but from user perspective, **data analyses tools roughly converge to a similar interface**.
- **Integration** between data analyses tools is best (by far) in **R**.
  - Dependency management and shared data structures.
- **R** has the most **advanced functionality for official statistics** (e.g. SAE, SDC,...)
- **R** has the **highest level of quality control** on extension packages.
- **Python** is **strongest in certain ML/DL techniques**. From a user perspective python feels heavily fragmented.
- **Julia** combines the best ideas of many years of technical computing.

The future?


Rpy2
reticulate
URSA LABS
JuliaCall
RCall
PyCall
python
julia
PyJulia

$[003]> Questions_

Thank you for your attention!