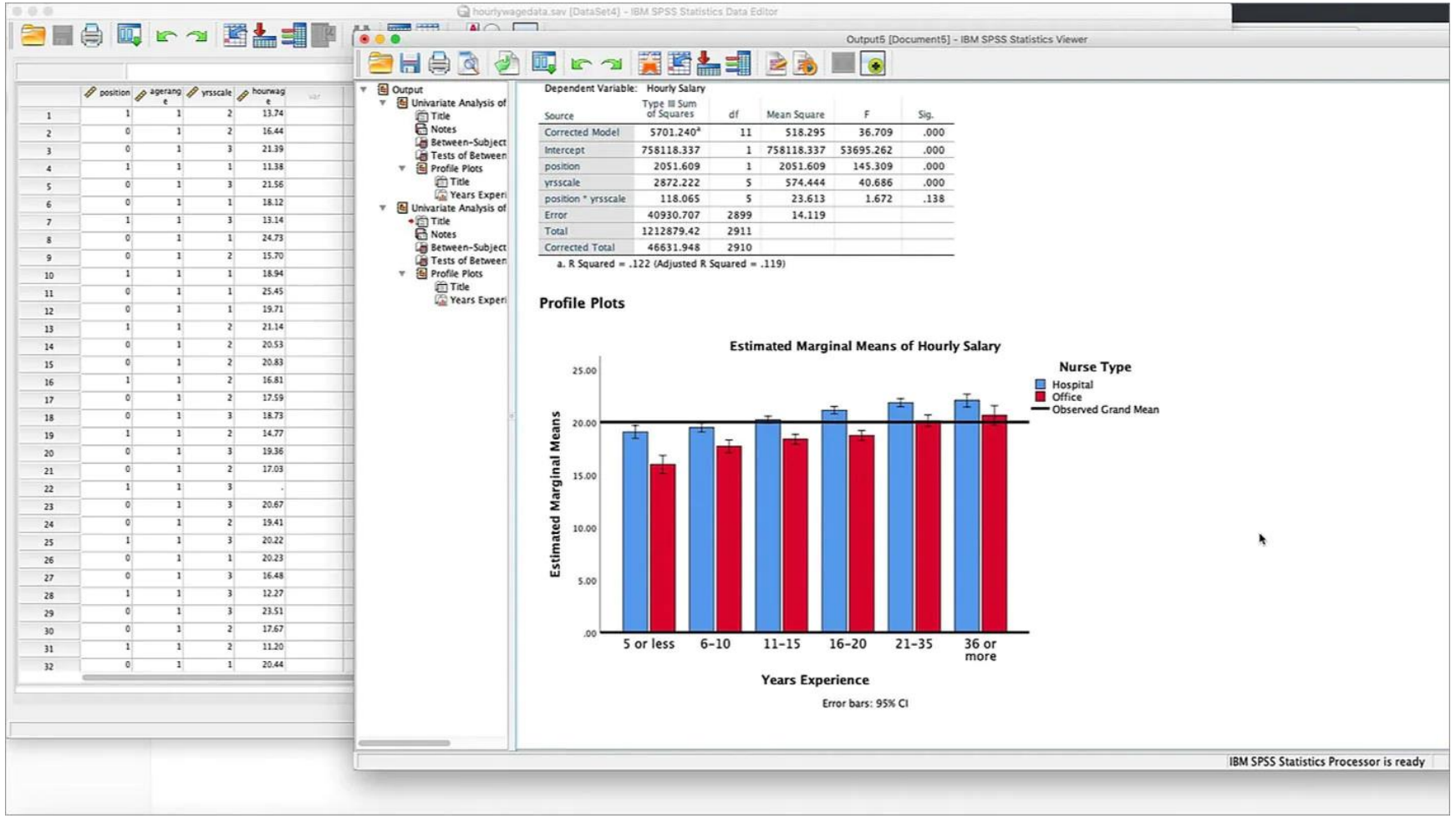
Julia

Tomas Fiers

Lunchtime data club at the School of Psychology

Dec 7, 2022

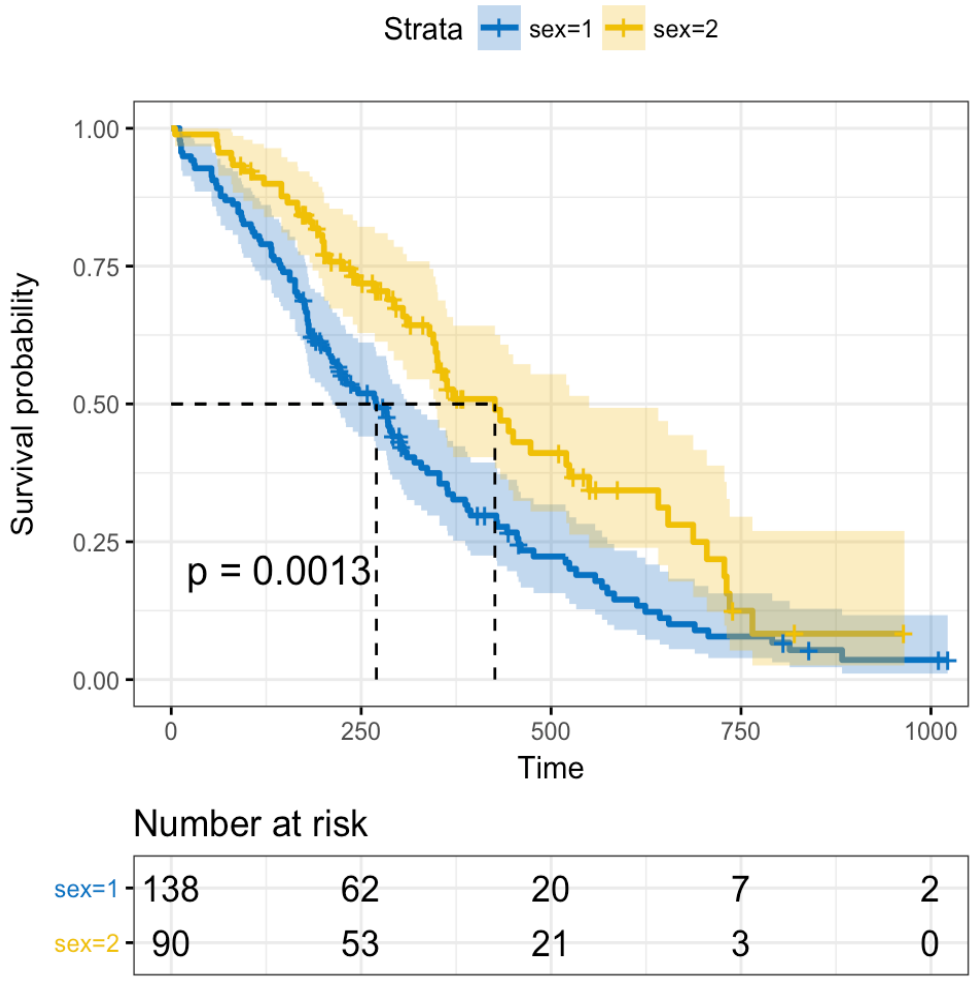
Why program?


hourlywagedata.sav [DataSet4] - IBM SPSS Statistics Data Editor
Output5 [Document5] - IBM SPSS Statistics Viewer
Output
Univariate Analysis of
Title
Notes
Between-Subject
Tests of Between
Profile Plots
Title
Years Experi
Univariate Analysis of
Title
Notes
Between-Subject
Tests of Between
Profile Plots
Title
Years Experi
Dependent Variable: Hourly Salary
Source | Type III Sum of Squares | df | Mean Square | F | Sig.
Corrected Model | 5701.240a | 11 | 518.295 | 36.709 | .000
Intercept | 758118.337 | 1 | 758118.337 | 53695.262 | .000
position | 2051.609 | 1 | 2051.609 | 145.309 | .000
yrsscale | 2872.222 | 5 | 574.444 | 40.686 | .000
position * yrsscale | 118.065 | 5 | 23.613 | 1.672 | .138
Error | 40930.707 | 2899 | 14.119
Total | 1212879.42 | 2911
Corrected Total | 46631.948 | 2910
a. R Squared = .122 (Adjusted R Squared = .119)
Profile Plots
Estimated Marginal Means of Hourly Salary
Nurse Type
Hospital
Office
Observed Grand Mean
Estimated Marginal Means
25.00
20.00
15.00
10.00
5.00
.00
5 or less
6-10
11-15
16-20
21-35
36 or more
Years Experience
Error bars: 95% CI
IBM SPSS Statistics Processor is ready

Why program?

- Automate analyses
  → less error-prone
- Customize
  - Custom plots
  - Custom analyses
- Run simulations!
- For fun & brain-exercise

An example custom plot



Choosing a programming language

	First release	Free & open? (hackable, "own your code runner")	Online community (Learning resources & documentation)
• Main choices			
R	1995 / 1976 (S)	Yes	Huge
Python	1991	Yes	Huge
• Others			
Julia	2012	Yes	Medium
Matlab	1979	No	Large

- ..is also choosing a community

Vic or
@vzverovich

Replying to @lefticus

Julia is Matlab without users

3:28 AM · Feb 21, 2022

Guillaume Dalle
@giomdal

Matlab is Julia without open source contributors 🤷

Vic or @vzverovich · Feb 21

Replying to @lefticus

Julia is Matlab without users

8:59 AM · Mar 20, 2022

Julia syntax

```julia
data = [5, 1, 8, 3]      # Some random data

function average(x)
    N = length(x)        # Number of elements
    total = 0            # Initialize sum
    for i in 1:N         # Loop
        total += x[i]    # Add the i'th element
    end
    return total / N     # Calculate mean
end

μ = average(x)
```

“Avoid for-loops”
“Write vectorized code”

Code compilation

- If **one line** of Julia code corresponds to just a few CPU instructions
- ..then the same line in base Python / R / Matlab will often correspond to an order of magnitude **more CPU instructions** *
  - ..That's why the code that does the 'real' numeric work in these languages is actually written in C / C++
    NumPy, PyTorch, Tensorflow, dplyr, ..: all have their core written in a different language
  - ..That's why, to have your program in these languages run fast, you're discouraged from writing for-loops for numeric code..
  - ..and instead use the provided library functions (e.g. np.quantile(…))
  - Python is often used as "glue-code" (see next slide)
  - If you want a custom numeric algorithm that's not provided by the libraries, you need to learn C/C++
    The "two languages-problem"

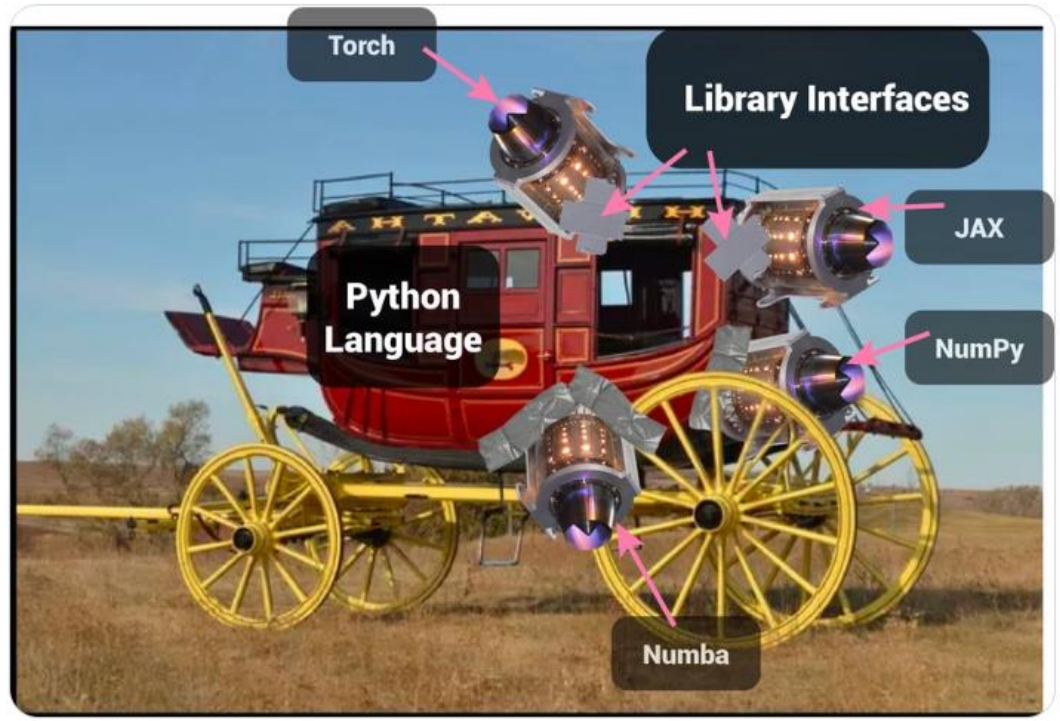
* Matlab added JIT compilation in 2015 (but it's rather opaque):
https://www.mathworks.com/products/matlab/performance.html

Python can have JIT compilation via the fantastic **Numba** package.
(But you can only use base Python with Numba, not arbitrary other packages).

Miles Cranmer
@MilesCranmer

The more I use Julia, the more Python and its numeric libraries look like a Victorian-era stagecoach with jet engines duct-taped to it, each pointing a different direction (=mutually incompatible).

It's such a weird ecosystem, and makes it so much harder for users to contribute.



5:50 PM · Nov 7, 2022

JIT compilation

- If **one line** of Julia code corresponds to just a few CPU instructions
- ..then the same line in base Python / R / Matlab* will often correspond to an order of magnitude **more CPU instructions**

- Why is this?
  - The same code (say, `out = x + y`) does different things, based on the **type** of x and y
    - If they're integers (8+3), use the `leaq` CPU instruction
    - If one is a float (8+3.3), call `convert` and use the math co-processor
    - If they're both *plots*, call subroutines, to compose them together into a bigger figure
    - …
  - Python, R, and Matlab need to check the types of x and y *every time the line is run,* and then call the appropriate subroutines
    - Hence all these extra CPU instructions
  - Julia will ***infer*** the types of x and y
    - When? The first time that the function which contains our code is called
    - It does this (i.e. 'type inference') based on the types of the objects the function was called with, and by analyzing the function source code
  - I then compiles a fast version of the function
    This is just-in-time (JIT) compilation

Data analysis in Julia

- DataFrames.jl
  - tidyverse's dplyr & Python's Pandas equivalent
- `missing` datatype is built-in
  - distinct from `nothing`
- I plot using Python's matplotlib 😁
  - Via PyPlot.jl
  - There's also Makie.jl
  - ..and Gadfly.jl, which is ggplot-inspired

Julia likes

- Unicode variable names & operators

```
izh() = begin
    # Conductance-based synaptic current
    I_syn = gₑ*(v-Eₑ) + gᵢ*(v-Eᵢ)
    # Izhikevich 2D system
    Δ.v = (k*(v-vₗ)*(v-vₜ) - u - I_syn) / C  # Membrane potential
    Δ.u = a*(b*(v-vᵣ) - u)                   # Adaptation current
    # Synaptic conductance decay
    # (gₑ is sum over all exc synapses)
    Δ.gₑ = -gₑ / τ
    Δ.gᵢ = -gᵢ / τ
end
has_spiked() = (v ≥ vₛ)
on_self_spike() = begin
    v = vᵣ
    u += Δu
end
```

Julia likes

- Community
  - Discourse forum & Slack
  - Scientists
  - Contribute to ecosystem (open source)
- As close-to-the-metal as you like
  - Look under the hood
  - Understand why something is slow / fast
- “structs and functions” design style
  - Versus when you’re writing software in Python, you tend to use OOP (inheritance)
- Inspectability
  - `@edit` to jump to source code of anything… amazing
  - `@code_native` to see cpu instructions
  - `?` for documentation
  - …
- Dependency management
  - Project.toml
  - Manifest.toml
- Macro’s
  - Lisp-like. ‘Code as data’

Julia annoyances

- **Package startup time** 💀 (“time-to-first-plot”)
  - Language developers are improving this atm
- No winning plotting package yet
- `name.<tab>` completion (API discovery) not as good as Python
- Getting floats to print with lower precision is way more difficult than it should be for new users
- Traits / interfaces (lack of)
- Error handling is underdeveloped / under practiced (→ silent fails & crashes)
- See also:
  - yuri.is/not-julia
  - danluu.com/julialang
  - viralinstruction.com/posts/badjulia

“Julia has a correctness problem”

- (i.e. there’s nasty hidden bugs everywhere)
- Not true for Base Julia:
  - every line there is pored over by many language developers
  - automatic test coverage is very comprehensive
- For other people’s packages:
  - Not a problem in my experience.
  - But you have to inspect the packages that you use, if they’re not in Julia Base; and make a value judgement about their quality
  - A lot of Julia packages are of *very* high quality in my experience
    - Except for the lack of error checking (of inputs and outputs)
      - Julia doesn’t hold your hand:
        you gotta know what you’re doing mathematically / numerically / statistically

Why do I use Julia?

- Advent of Code :) (2021)
- Physical units in neuron simulations

- I could keep using
  - my Jupyter notebook workflow
  - Matplotlib

```
parameters = (
    # Izhikevich neuron
    C  =  100    * pF
    k  =    0.7  * (nS/mV)
    vₗ = - 60    * mV
    vₜ = - 40    * mV
    a  =    0.03 / ms
    b  = -  2    * nS
    vₛ =   35    * mV
    vᵣ = - 50    * mV
    Δu =  100    * pA
    # Synapses
    Eₑ =   0 * mV
    Eᵢ = -80 * mV
    τ  =   7 * ms
    # Inputs
    Nₑ = 40
    Nᵢ = 10
    N = Nₑ + Nᵢ
    Δgₑ = 60nS / Nₑ
    Δgᵢ = 60nS / Nᵢ
    # Integration
    Δt = 0.1ms
    T  = 10seconds
)
```

Julia tips

- Code must be type-inferable ("type-stable")
  - Put everything in (small) functions
  - If using globals: `const`, or typed
- Read the manual
  - Especially the "Performance tips" section, if you're wondering why your code is not as fast as promised. Also:
- **Ask questions on the forum**
  - discourse.julialang.org
- Use Revise.jl (Use all of Tim Holy's packages actually). Plus:
  - If using VS Code, there's a plugin for Julia
  - On Windows, use the Julia REPL in the Windows Terminal
  - Checkout startup.jl
- Don't load unnecessary packages

Should you use julia ?

- Do you ‘just’ need data analysis, automation, and pretty, customized plots?
  - Then, no

- Or do you also write custom numeric algorithms / simulations?
  - Then, yes :)
  - ..Unless you already know Matlab and don’t have the time
  - ..Plus, Python and R have huge ecosystems of packages that might already do your custom thing
    - Also, Python has **Numba** for JIT-optimization of hot inner loops (numba.pydata.org). That might be enough

Code sharing, git, GitHub

- tfiers.github.io/phd
  - made with JupyterBook
  - auto-built and -published with GitHub Actions on GitHub Pages

- github.com/schluppeck/ng-data-club
  - /presentations subdirectory, by ISO8601 date+slug