JULIA

NOTES FROM THE FRONT LINES

MassMutual Data Science Seminar

April 2018

by Andy Reagan

DISCLAIMER

This is mostly my opinion.

JULIA PROJECT GOALS

(WHY IS THIS LANGUAGE AROUND)

- built by hackers
- speed: less clunkly + faster than R, faster than MATLAB
- dynamic: run and test code in realtime

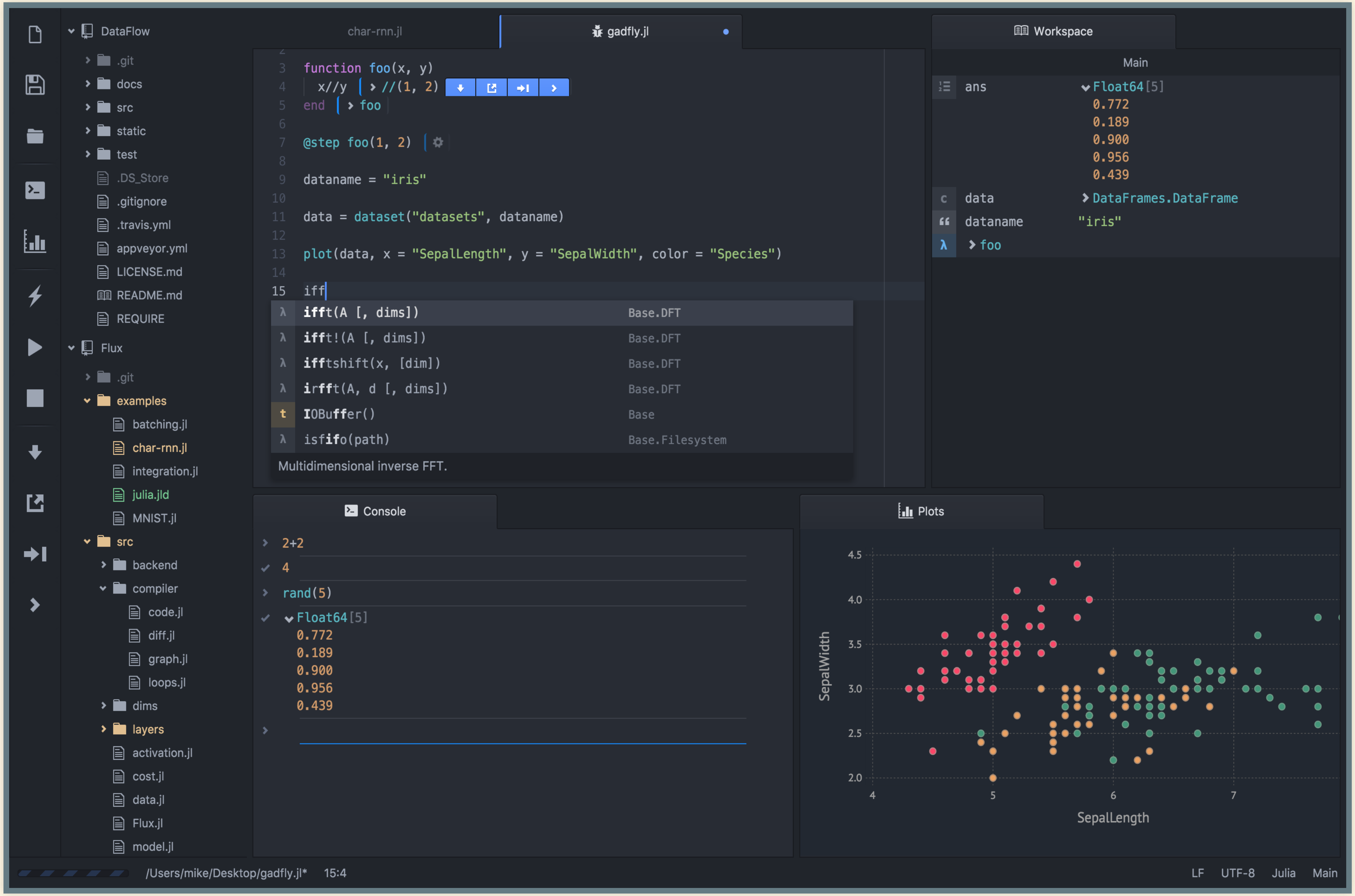
WORKFLOW

Writes like R/MATLAB, code it like Python

- JUNO IDE (bundled into JuliaPro)
- Remote kernel hydrogen
  - Like any other kernel
- Jupyter notebook/lab (same)
- REPL (like IPython, R console)
- Spark.jl (think: PySpark or RSpark)
  - https://github.com/dfdx/Spark.jl

JULIAPRO IDE

Feel at home coming from RStudio/MATLAB IDEs.



```
function foo(x, y)
  x//y  > //(1, 2)
end  > foo

@step foo(1, 2)

dataname = "iris"

data = dataset("datasets", dataname)

plot(data, x = "SepalLength", y = "SepalWidth", color = "Species")

iff
```

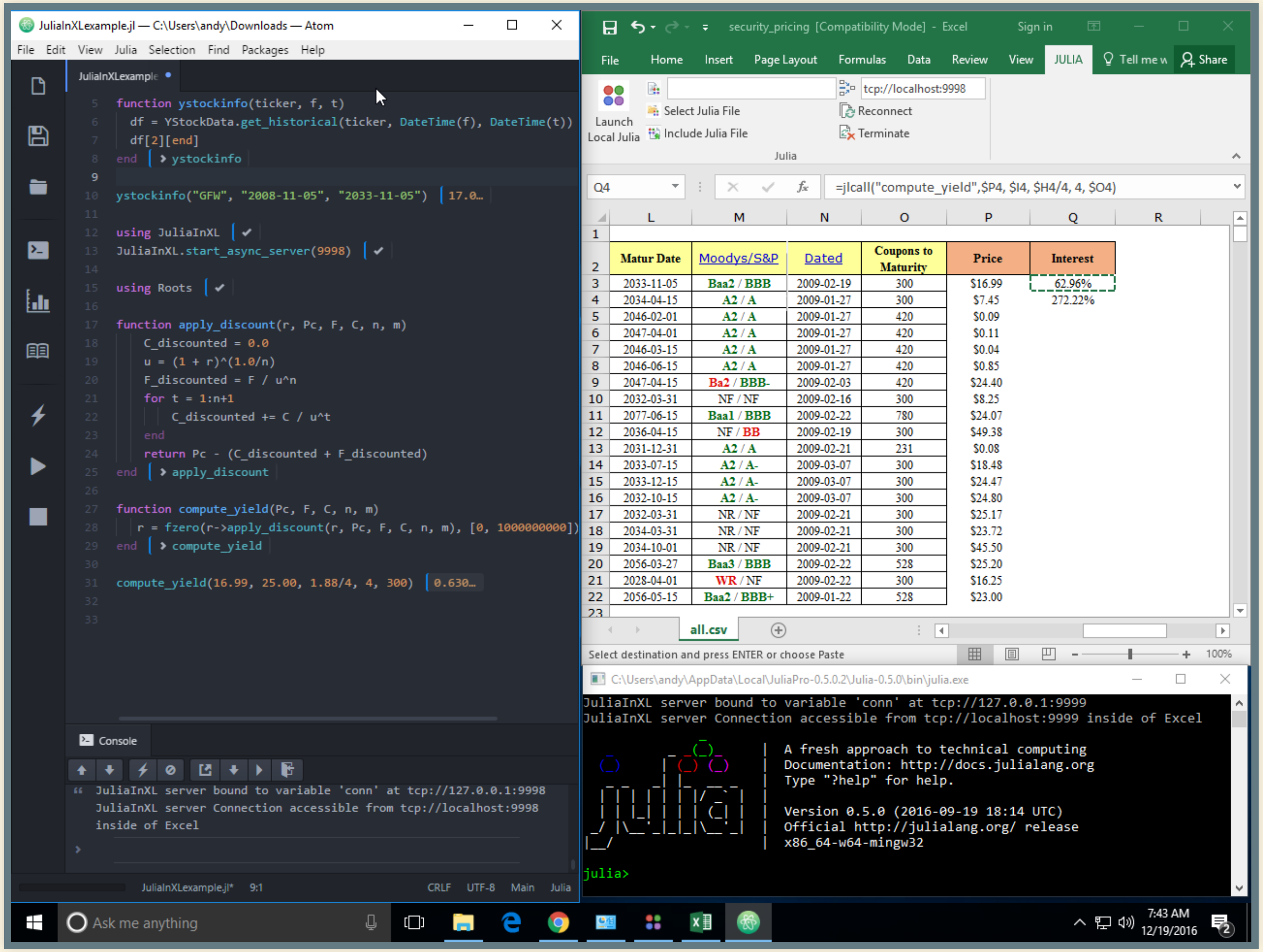
```
λ ifft(A [, dims])          Base.DFT
λ ifft!(A [, dims])         Base.DFT
λ ifftshift(x, [dim])       Base.DFT
λ irfft(A, d [, dims])      Base.DFT
t IOBuffer()                Base
λ isfifo(path)              Base.Filesystem
```

Multidimensional inverse FFT.

```
ans         Float64[5]
              0.772
              0.189
              0.900
              0.956
              0.439
data        DataFrames.DataFrame
dataname    "iris"
foo
```

```
> 2+2
✓ 4
> rand(5)
✓ Float64[5]
   0.772
   0.189
   0.900
   0.956
   0.439
>
```

IT RUNS IN EXCEL...


JuliaInXLexample.jl — C:\Users\andy\Downloads — Atom
File Edit View Julia Selection Find Packages Help
JuliaInXLexample
security_pricing [Compatibility Mode] - Excel
Sign in
File Home Insert Page Layout Formulas Data Review View JULIA Tell me w Share
Launch Local Julia
Select Julia File
Include Julia File
tcp://localhost:9998
Reconnect
Terminate
Julia
Q4
all.csv
Console
JuliaInXLexample.jl* 9:1 CRLF UTF-8 Main Julia
Ask me anything
7:43 AM
12/19/2016

```julia
function ystockinfo(ticker, f, t)
  df = YStockData.get_historical(ticker, DateTime(f), DateTime(t))
  df[2][end]
end

ystockinfo("GFW", "2008-11-05", "2033-11-05")   17.0…

using JuliaInXL
JuliaInXL.start_async_server(9998)

using Roots

function apply_discount(r, Pc, F, C, n, m)
    C_discounted = 0.0
    u = (1 + r)^(1.0/n)
    F_discounted = F / u^n
    for t = 1:n+1
        C_discounted += C / u^t
    end
    return Pc - (C_discounted + F_discounted)
end

function compute_yield(Pc, F, C, n, m)
   r = fzero(r->apply_discount(r, Pc, F, C, n, m), [0, 1000000000])
end

compute_yield(16.99, 25.00, 1.88/4, 4, 300)   0.630…
```

```
JuliaInXL server bound to variable 'conn' at tcp://127.0.0.1:9998
JuliaInXL server Connection accessible from tcp://localhost:9998
inside of Excel
```

`=jlcall("compute_yield",$P4, $I4, $H4/4, 4, $O4)`

Matur Date	Moodys/S&P	Dated	Coupons to Maturity	Price	Interest
2033-11-05	Baa2 / BBB	2009-02-19	300	$16.99	62.96%
2034-04-15	A2 / A	2009-01-27	300	$7.45	272.22%
2046-02-01	A2 / A	2009-01-27	420	$0.09	
2047-04-01	A2 / A	2009-01-27	420	$0.11	
2046-03-15	A2 / A	2009-01-27	420	$0.04	
2046-06-15	A2 / A	2009-01-27	420	$0.85	
2047-04-15	Ba2 / BBB-	2009-02-03	420	$24.40	
2032-03-31	NF / NF	2009-02-16	300	$8.25	
2077-06-15	Baa1 / BBB	2009-02-22	780	$24.07	
2036-04-15	NF / BB	2009-02-19	300	$49.38	
2031-12-31	A2 / A	2009-02-21	231	$0.08	
2033-07-15	A2 / A-	2009-03-07	300	$18.48	
2033-12-15	A2 / A-	2009-03-07	300	$24.47	
2032-10-15	A2 / A-	2009-03-07	300	$24.80	
2032-03-31	NR / NF	2009-02-21	300	$25.17	
2034-03-31	NR / NF	2009-02-21	300	$23.72	
2034-10-01	NR / NF	2009-02-21	300	$45.50	
2056-03-27	Baa3 / BBB	2009-02-22	528	$25.20	
2028-04-01	WR / NF	2009-02-22	300	$16.25	
2056-05-15	Baa2 / BBB+	2009-01-22	528	$23.00	

Select destination and press ENTER or choose Paste

C:\Users\andy\AppData\Local\JuliaPro-0.5.0.2\Julia-0.5.0\bin\julia.exe

```
JuliaInXL server bound to variable 'conn' at tcp://127.0.0.1:9999
JuliaInXL server Connection accessible from tcp://localhost:9999 inside of Excel

               _
   _       _ _(_)_     |  A fresh approach to technical computing
  (_)     | (_) (_)    |  Documentation: http://docs.julialang.org
   _ _   _| |_  __ _   |  Type "?help" for help.
  | | | | | | |/ _` |  |
  | | |_| | | | (_| |  |  Version 0.5.0 (2016-09-19 18:14 UTC)
 _/ |\__'_|_|_|\__'_|  |  Official http://julialang.org/ release
|__/                   |  x86_64-w64-mingw32

julia>
```

GETTING UP AND RUNNING

Download binary from here:

https://julialang.org/downloads/

CORE PACKAGES

General programming

DataStructures

LightGraphs

General Math

Calculus

DataFrames

StatsBase

Distributions

HypothesisTests

GLM

Optimization

JuMP

Optim

Roots

Databases

ODBC

JDBC

Building UIs and Visualization

Gadfly

PyPlot

Interact

Deep Learning and Machine Learning

Knet

Clustering

DecisionTree

Interoperability with other languages

RCall

JavaCall

PyCall

File and data formats

JSON

HDF5

JLD

Economics and Finance

QuantEcon

FUN/BORING* STUFF

- data type system (typing optional)
- JIT compilation
- next to assembly
- multiple dispatch
- homoiconicity
- *only depends on perspective

DATA TYPES

What you might expect...

- Functions
- Strings (unicode)
- Numbers
- Arrays
- Matrices
- Sparse versions

Define your own!

Some choices: Int8, UInt8, Int16, UInt16, Int32, UInt32, Int64, UInt64, Int128, UInt128, Float16, Float32, and Float64

```
local x::Int8  # in a local declaration
x::Int8 = 10   # as the left-hand side of an assignment
```

```
abstract type Number end
abstract type Real     <: Number end
abstract type AbstractFloat <: Real end
abstract type Integer  <: Real end
abstract type Signed   <: Integer end
abstract type Unsigned <: Integer end
```

ROLL YOU OWN

```
julia> struct Foo
           bar
           baz::Int
           qux::Float64
       end
```

```
julia> foo = Foo("Hello, world.", 23, 1.5)
Foo("Hello, world.", 23, 1.5)

julia> typeof(foo)
Foo
```

ASSEMBLY

https://docs.julialang.org/en/stable/stdlib/base/#Base.co

```
julia> code_native(+,(Int64,Int64))
    .section    __TEXT,__text,regular,pure_instructions
Filename: int.jl
    pushq    %rbp
    movq    %rsp, %rbp
Source line: 32
    leaq    (%rdi,%rsi), %rax
    popq    %rbp
    retq
Source line: 32
    nopw    (%rax,%rax)
```

```
julia> code_native(/,(Int64,Int64))
    .section    __TEXT,__text,regular,pure_instructions
Filename: int.jl
    pushq    %rbp
    movq    %rsp, %rbp
Source line: 38
    xorps    %xmm0, %xmm0
    cvtsi2sdq    %rdi, %xmm0
    xorps    %xmm1, %xmm1
    cvtsi2sdq    %rsi, %xmm1
    divsd    %xmm1, %xmm0
    popq    %rbp
    retq
Source line: 38
    nopw    (%rax,%rax)
```

PARALLELIZATION

```
a = SharedArray{Float64}(10)
@parallel for i = 1:10
    a[i] = i
end
```

MULTIPLE DISPATCH

Functions have multiple methods depending upon args.

```
julia> code_native(+,(Char,Int64))
    .section    __TEXT,__text,regular,pure_instructions
Filename: char.jl
    pushq    %rbp
    movq    %rsp, %rbp
Source line: 40
    testl    %edi, %edi
    js    L24
    movslq    %esi, %rax
    cmpq    %rsi, %rax
    jne    L39
Source line: 4
    addl    %edi, %esi
    js    L54
Source line: 40
    movl    %esi, %eax
```

PACKAGE MANAGEMENT

So, so cool. Whiteboard!

METAPROGRAMMING

Rejoice, this is how it should work!

> *Every Julia program starts life as a string.*

```
julia> prog = "1 + 1"
"1 + 1"
```

```
julia> ex1 = parse(prog)
:(1 + 1)

julia> typeof(ex1)
Expr
```

```
julia> ex1.args
3-element Array{Any,1}:
  :+
 1
 1
```

```
julia> ex2 = Expr(:call, :+, 1, 1)
:(1 + 1)
```

```
julia> ex1 == ex2
true
```

QUOTING

```
julia>      :(a + b*c + 1)  ==
       parse("a + b*c + 1") ==
       Expr(:call, :+, :a, Expr(:call, :*, :b, :c), 1)
true
```

FINALLY, EVAL

```
julia> prog = "1 + 1"
"1 + 1"

julia> parse(ans)
:(1 + 1)

julia> eval(ans)
2

julia> typeof(eval(ans))
Int64

julia> typeof(eval(:(1+1)))
Int64
```

THAT WAS FUN!

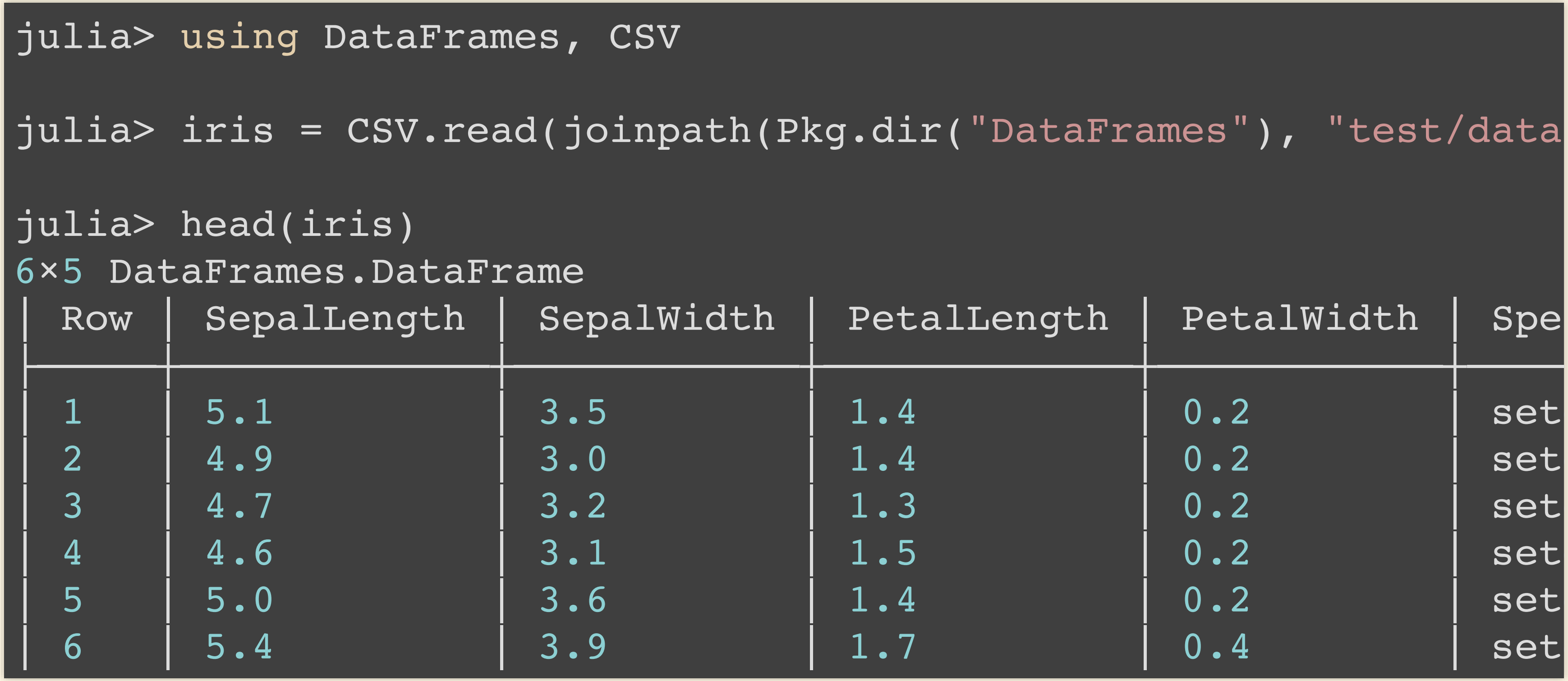
DS WITH JULIA

Ingredients:

- Data
- Model fitting

DATAFRAMES.JL

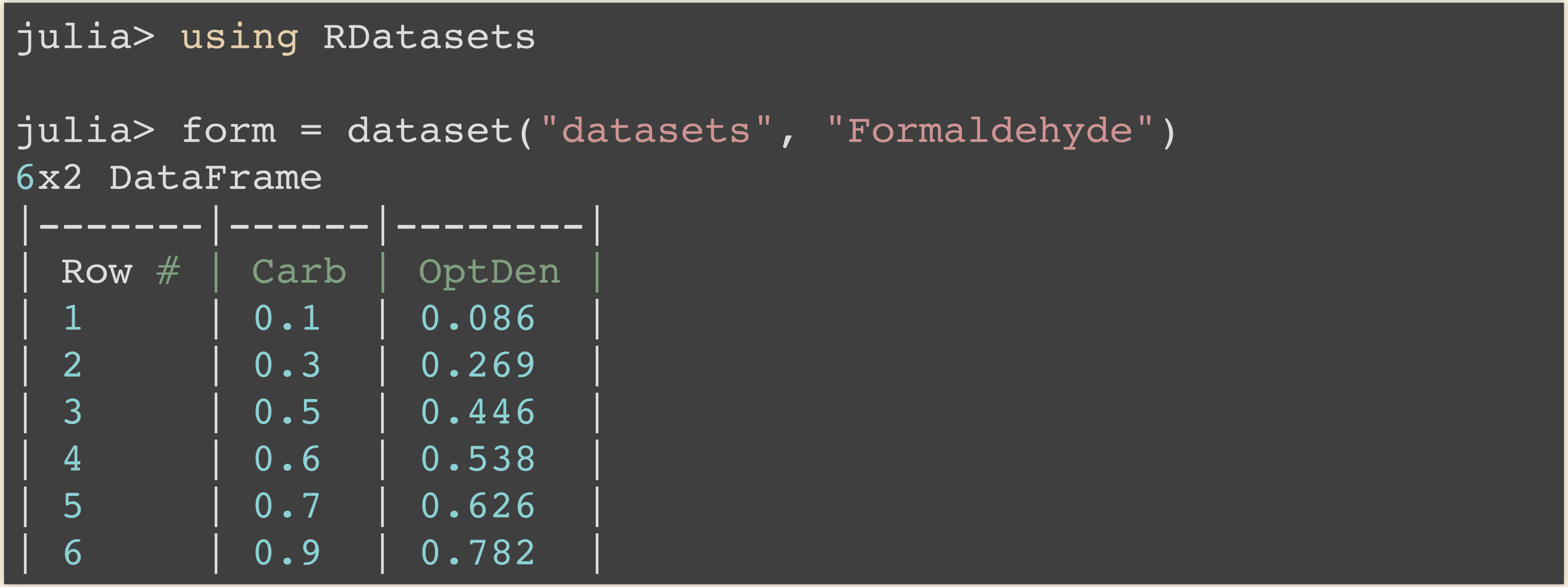
```
julia> data = DataFrame(X=[1,2,3], Y=[2,4,7])
3x2 DataFrame
|--------|---|---|
| Row #  | X | Y |
| 1      | 1 | 2 |
| 2      | 2 | 4 |
| 3      | 3 | 7 |
```



```
julia> using DataFrames, CSV

julia> iris = CSV.read(joinpath(Pkg.dir("DataFrames"), "test/data

julia> head(iris)
6×5 DataFrames.DataFrame
│ Row │ SepalLength │ SepalWidth │ PetalLength │ PetalWidth │ Spe
├─────┼─────────────┼────────────┼─────────────┼────────────┼────
│ 1   │ 5.1         │ 3.5        │ 1.4         │ 0.2        │ set
│ 2   │ 4.9         │ 3.0        │ 1.4         │ 0.2        │ set
│ 3   │ 4.7         │ 3.2        │ 1.3         │ 0.2        │ set
│ 4   │ 4.6         │ 3.1        │ 1.5         │ 0.2        │ set
│ 5   │ 5.0         │ 3.6        │ 1.4         │ 0.2        │ set
│ 6   │ 5.4         │ 3.9        │ 1.7         │ 0.4        │ set
```



```
julia> using RDatasets

julia> form = dataset("datasets", "Formaldehyde")
6x2 DataFrame
|--------|-------|---------|
| Row #  | Carb  | OptDen  |
| 1      | 0.1   | 0.086   |
| 2      | 0.3   | 0.269   |
| 3      | 0.5   | 0.446   |
| 4      | 0.6   | 0.538   |
| 5      | 0.7   | 0.626   |
| 6      | 0.9   | 0.782   |
```

Also, `read_rda` function to read directly.

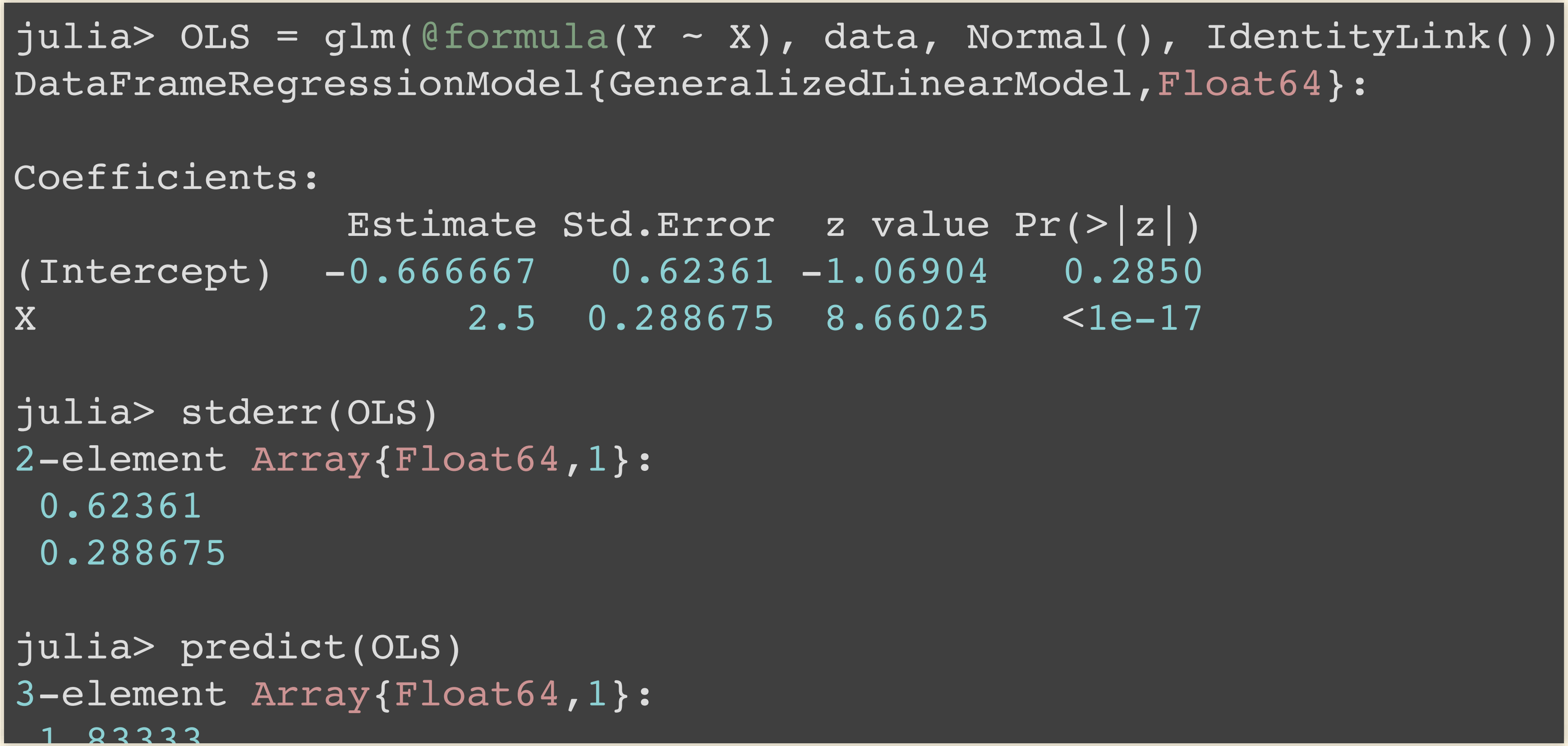
FUNCTIONALITY

- `size()`,`head()`,`tail()`
- `nrow()`,`ncol()`,`length()`
- `describe()`
- `showcols()`
- `names()`,`eltypes()`,`names!()`
- `Missing`
- `merge!()`,`hcat()`,`insert!()`...
- `map`,`groupby`
- etc...

MODELING

- GLM: GLM.jl
- MixedModels: MixedModels.jl
- Decision Trees: DecisionTree.jl (yes, boosting too)
- Deep Learning: Knet, Flux, Mocha

Quick example next.



```
julia> OLS = glm(@formula(Y ~ X), data, Normal(), IdentityLink())
DataFrameRegressionModel{GeneralizedLinearModel,Float64}:

Coefficients:
              Estimate Std.Error  z value Pr(>|z|)
(Intercept)  -0.666667   0.62361 -1.06904   0.2850
X                  2.5  0.288675  8.66025   <1e-17

julia> stderr(OLS)
2-element Array{Float64,1}:
 0.62361
 0.288675

julia> predict(OLS)
3-element Array{Float64,1}:
 1.83333
```

MAJOR WEAKNESSES

Weak spot for me: dataframes.jl clunkier than Pandas, much clunkier than dplyr...

TAKEAWAY

Just so much fun.

Not a primary language for data manipulation/exploration for me, but can build models fast!

RESOURCES

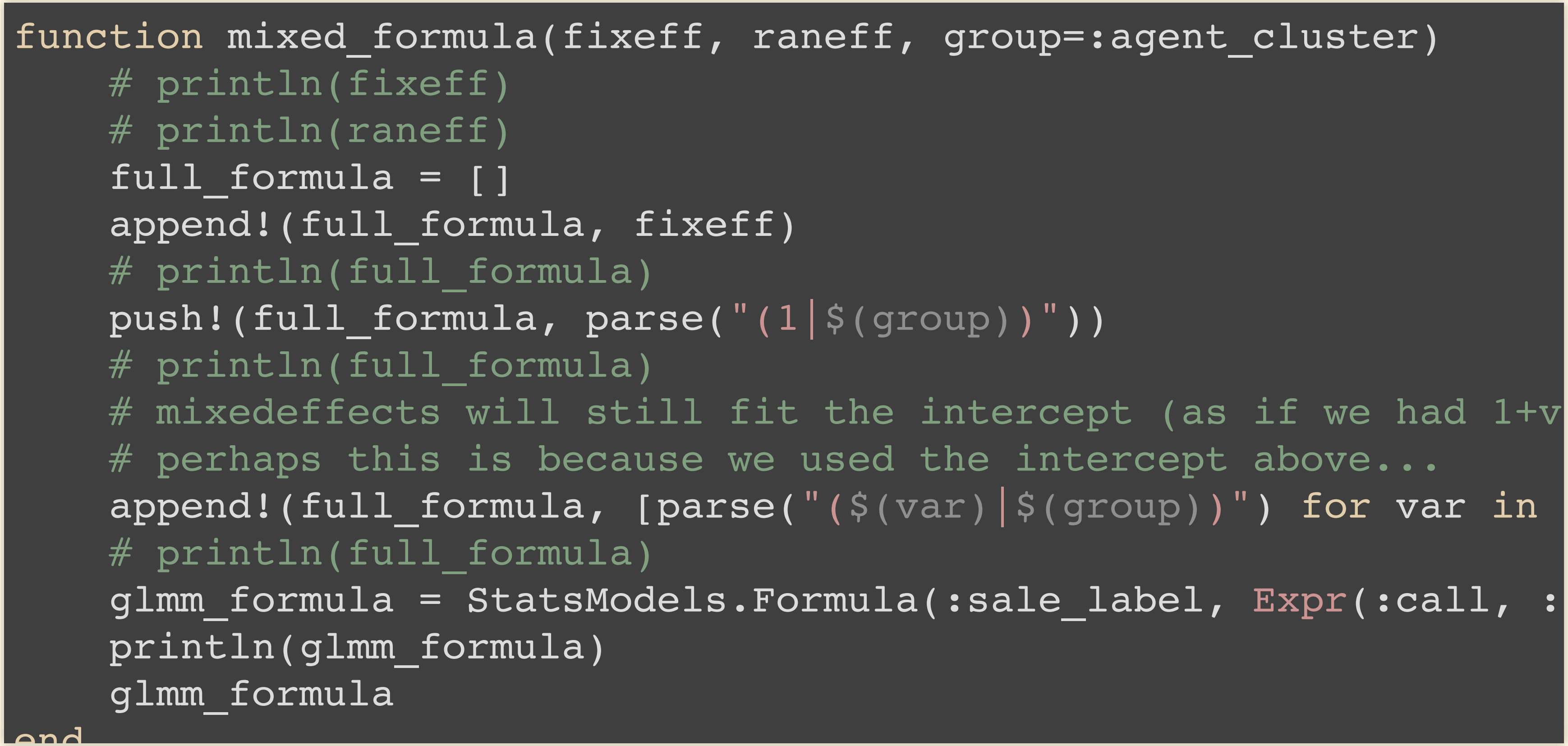
- Julia and IJulia cheatsheet
- Learn Julia in a few minutes
- Learn Julia the Hard Way
- Julia by Example
- Hands-on Julia

RESOURCES (CONT'D)

- Tutorial for Homer Reid's numerical analysis class
- An introductory presentation
- Videos from the Julia tutorial at MIT
- YouTube videos from the JuliaCons

QUESTIONS?

Give it a try!



```
function mixed_formula(fixeff, raneff, group=:agent_cluster)
    # println(fixeff)
    # println(raneff)
    full_formula = []
    append!(full_formula, fixeff)
    # println(full_formula)
    push!(full_formula, parse("(1|$(group))"))
    # println(full_formula)
    # mixedeffects will still fit the intercept (as if we had 1+v
    # perhaps this is because we used the intercept above...
    append!(full_formula, [parse("($(var)|$(group))") for var in
    # println(full_formula)
    glmm_formula = StatsModels.Formula(:sale_label, Expr(:call, :
    println(glmm_formula)
    glmm_formula
end
```

I wrote a package a long time ago...

https://github.com/JuliaLang/METADATA.jl/tree/metada
v2/OpenFOAM/versions/0.0.1

https://github.com/andyreagan/OpenFOAM.jl

Boundary stuff with Julia

Knet

Flux

CUDAnative

DataFlow

JuliaDiff