juliacon

Kezdi.jl: A data analysis package for economists

Miklós Koren[1, 2, 3, 4] and Gergely Attila Kiss[1, 2]

[1]Central European University, Vienna, Austria
[2]HUN-REN KRTK, Budapest, Hungary
[3]CEPR, London, United Kingdom
[4]CESifo, Munich, Germany

ABSTRACT

Economists overwhelmingly rely on proprietary data analysis languages such as Stata and MATLAB for their research computing needs. The transition to open-source languages like Julia presents various challenges due to differences in syntax, functionality, and best practices. We introduce `Kezdi.jl`, a data analysis package designed for economists that provides a Stata-like interface for working with data frames in Julia. The package is built on `DataFrames.jl` and related libraries, but uses a streamlined macro-based interface to eliminate common points of confusion. By emulating best practices from Stata, `Kezdi.jl` allows economists to be productive in Julia from day one. It supports a wide range of data wrangling and analysis tasks, including cleaning and transforming data, handling missing values, generating new variables, aggregating data, and running regressions.

Keywords

Julia, Data analysis, Data wrangling, Stata

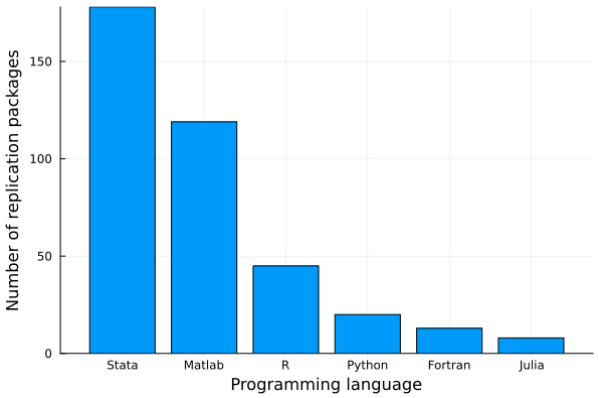

Fig. 1: Programming language usage in economics replication packages submitted to the Review of Economic Studies. A package can contain code in multiple languages.

1. Introduction

Research computing is central to the scientific workflow of economists. When designing a new research software tool, it is important to understand the needs and preferences of the target audience. We studied 245 replication packages submitted to the Review of Economic Studies between October 2020 and November 2024 to understand what programming languages economists use and how.[1] Figure 1 shows the number of packages that contain code in various languages. Stata is by far the most popular, used in 73% of packages, followed by MATLAB with 49%. In contrast, only 3% of replication packages contain any Julia code, which is even less prevalent than Fortran.

Stata and MATLAB are often used together in the same replication package, with Stata focusing on data manipulation and analysis, while MATLAB handles computationally intensive tasks like model simulations. These findings highlight both the current dominance of proprietary scientific computing languages in economics and the large untapped potential for open source tools like Julia. Lowering the barriers to entry through familiar interfaces, as Kezdi.jl aims to do for Stata users, could play an important role in accelerating adoption.

Stata in particular has a dominant position in the data analysis and statistical modeling stages of the research workflow. The vast majority of code in a typical replication package with Stata scripts are devoted to data wrangling and related programming tasks. This is no surprise, given that a typical empirical economics paper assembles and harmonizes data from several disparate sources and the numerous judgment calls that go into this process [6].

The transition from proprietary languages like Stata or MATLAB to open-source ones like Julia presents several challenges. First, the syntax and mental model is very different. Stata is a data-centric language, where commands operate directly on columns of a dataset, and the details of the type system are hidden from the user. Second, Stata has had decades to build a rich library of statistical and econometric routines that are part of the core language. Third, they have developed coding styles and best practices that are familiar with economists. To name a concrete example, in Stata one can calculate a heteroskedasticity-robust standard error in a regression by simply adding the option `robust` to the `regress` command. Achieving the same in Julia requires selecting the appropriate regression function from a package like `GLM.jl`, exploring the documentation, and, finally, passing the correct arguments.

[1]These packages are available at `https://zenodo.org/communities/restud-replication`. Code to reproduce the analysis is available at `https://github.com/codedthinking/Kezdi.jl/blob/paper/paper/main.jl`.

`Kezdi.jl` [2] is a data analysis package for economists that aims to ease the transition to Julia by providing a Stata-like user experience. It allows users to manipulate data frames and conduct statistical analysis using Stata-inspired macro commands that are both powerful and easy to read.[3]

2. Key features of `Kezdi.jl`

The overarching design principle of `Kezdi.jl` is to provide a convenient and familiar interface for economists working with data frames. Here we highlight some key features that help achieve this.

2.1 Stata-like command syntax

```
@use "trade.dta"
@generate log_trade = log(trade)
@regress log_trade log_distance, robust
```

Fig. 2: Commands in Kezdi.jl follow Stata naming conventions and syntax. All commands start with the `@` sign. Options are separated from arguments by a comma.

`Kezdi.jl` has a good coverage of Stata's data manipulation commands, including `egen`, `collapse`, `reshape` and `mvencode`. Combined with Julia's superior performance, this allows economists to easily translate their Stata workflows. Some examples of common data wrangling tasks in Kezdi.jl:

```
@generate y = cond(x > 10, 1, 0)
@rename oldname newname
@mvencode y1 y2 y3 @if x < 0, mv(999)
@collapse avg_x = mean(x), by(group)
@reshape long y, i(id) j(year)
```

Fig. 3: Some data wrangling commands illustrating the syntax of Kezdi.jl.

2.2 Automatic column selection and vectorization

Variable names are automatically matched to DataFrame column names in `Kezdi.jl`. In standard Julia one would need to refer DataFrame columns with `df[:x]` or `df.x`. In other helper packages like `DataFramesMeta.jl` [9], one would need to refer to columns with symbols or strings.
Function calls are also automatically vectorized. In the examples above, `log(trade)` is equivalent to `log.(trade)` and `cond(x > 10, 1, 0)` is equivalent to `cond.(x .> 10, 1, 0)` in base Julia. The `cond` function returns 1 if the condition is true and 0 otherwise. This is similar to Stata's `cond()` function. Users can stop vectorization by adding a `~` before the function name, like `~log(trade)`.

Note that not all functions are vectorized. Statistical aggregation functions like `mean()` and `sum()` operate on the entire column by default and not elementwise. Any function that takes a `Vector` as its first argument is not vectorized by default. Users can manually vectorize these functions by adding a dot after the function name. This way the user has full control over the vectorization of their code. We believe these sensible default behaviors allow for concise and readable code.

2.3 Use any Julia function

Because `Kezdi.jl` supports arbitrary Julia syntax within commands, user-defined and external package functions work seamlessly with DataFrame columns.

```
using Dates
@generate date = Date(year, month, day)

using Statistics
@collapse std_x = std(x), by(group)
```

Fig. 4: Any existing or user-defined function can be used in `Kezdi.jl` code. These are also vectorized automatically.

This allows economists to easily extend their data manipulation and modeling capabilities beyond Stata's built-in functions by leveraging Julia's package ecosystem. Users can also write their own functions in Julia and use them in `Kezdi.jl` commands.

2.4 Handling of missing values

Missing values are a common pain point in data analysis. Julia requires users to handle missing values explicitly, which can be cumbersome, even if it less error-prone. Stata has special rules for the propagation of missing values. For example, `sum(x)` in Stata returns the sum of non-missing values of `x`. This is less explicit but more convenient for users.
We follows these rules to handle missing values:

(1) Mathematical operations return `missing` if any of their arguments are missing. This is also true for functions that do not support missing values. For example, `log(missing)` returns `missing`.
(2) Aggregation functions skip missing values by default. For example, `sum(x)` returns the sum of non-missing values of `x`.
(3) In logical expressions, `missing` is treated as `false`. This is unlike Stata, where `missing` is treated as `true`. This is a conscious decision to avoid common pitfalls with missing values.

```
using Kezdi
df = DataFrame(x=[1, 2, missing], y=[missing, 3, 4])
@collapse sum_x = sum(x)
# returns 3

@generate z = x + y
# returns [missing, 5, missing]
```

Fig. 5: Missing values are handled in a logical way.

[2] `https://docs.koren.dev/Kezdi.jl/`
[3] Stata® is a registered trademark of StataCorp LLC. This package implements command syntax similar to Stata® but is not affiliated with, endorsed by, or sponsored by StataCorp LLC. Any reference to Stata® is for informational purposes only and does not imply any relationship with or endorsement by StataCorp LLC.

2.5 Syntactic sugar for operating on a subset of rows

One of the most convenient features of Stata is the ability to restrict any command to a subset of rows using logical expressions with the `if` qualifier. `Kezdi.jl` implements the same with the `@if` macro.[4]

```
@replace trade = 0 @if ismissing(trade)
@regress log_trade log_distance @if trade > 0
@keep @if !ismissing(trade)
@collapse mean_x = mean(x) @if group ==
"treatment"
```

Fig. 6: The @if macro can be used to limit the scope of any command.

2.6 Integration with Julia's package ecosystem

`Kezdi.jl` builds on the `DataFrames.jl` ecosystem [8] to provide its data manipulation features. It also integrates tightly with other packages:

(1) `FreqTables.jl` [3] for frequency tables via `@tabulate`
(2) `FixedEffectModels.jl` [5] for estimating regressions with `@regress`
(3) `Chain.jl` [7] for piping a sequence of commands with `@with`
(4) `ReadStatTables.jl` [4] for importing Stata `.dta` files

3. Comparison with related packages

`Kezdi.jl` is not the first attempt to provide a simplified data analysis workflow in Julia. `DataFramesMeta.jl` [9] extends the DataFrame interface with easy-to-use macros. On the R side, dplyr [11] has been a hugely successful abstraction. `TidierData.jl` [10] ports many ideas from dplyr to Julia and served as a primary inspiration for many of our design choices.
texttttDouglass.jl is a related package that also aims to provide a Stata-like user experience in Julia [2]. The two packages share similar goals, but `Kezdi.jl` has a more extensive feature set and is more actively developed.

4. Conclusion

We introduced `Kezdi.jl`, a data analysis package for economists that provides a Stata-like interface to Julia's data manipulation and statistics ecosystem. `Kezdi.jl` aims to shorten the learning curve for economists transitioning from Stata, while also unlocking the power and expressiveness of Julia.
With `Kezdi.jl`, economists can manipulate data frames, calculate summary statistics, and estimate regressions using familiar keystrokes and mental models. At the same time, the package opens the door to Julia's rich ecosystem of packages for optimization, machine learning, visualization, and high-performance computing.
Our hope is that `Kezdi.jl` will accelerate adoption of Julia in economics and other social sciences, and ultimately lead to better, more reproducible research. We invite researchers and developers to try out the package and share their feedback on how we can expand its capabilities to better meet their scientific computing needs.

[4]Even though `@if` looks like a macro, it is not implemented as one, because `if` is a reserved word in Julia. The macro call is processed during parsing the `Kezdi.jl` command.

Acknowledgments

The package is named in memory of Gábor Kézdi[5], a prominent applied microeconomist and advocate of democratizing data analysis tools [1]. We are grateful to the Julia community for their support and feedback, and to the Review of Economic Studies for hosting the replication packages that inspired this work.

5. References

[1] G. Békés and G. Kézdi. *Data Analysis for Business, Economics, and Policy*. Cambridge University Press, 2021.
[2] Johannes Boehm. Douglass.jl [software], 202-. https://github.com/jbboehm/Douglass.jl.
[3] Milan Bouchet-Valat et al. FreqTables.jl [software], 2014. https://github.com/nalimilan/FreqTables.jl.
[4] Junyuan Chen et al. ReadStatTables.jl [software], 2021. https://github.com/junyuan-chen/ReadStatTables.jl.
[5] Matthieu Gomez et al. FixedEffectModels.jl [software], 2023. https://github.com/FixedEffects/FixedEffectModels.jl.
[6] Nick Huntington-Klein, Andreu Arenas, Emily Beam, Marco Bertoni, Jeffrey R Bloem, Pralhad Burli, Naibin Chen, et al. The influence of hidden researcher decisions in applied microeconomics. *Economic Inquiry*, 59(3):944–960, 2021.
[7] Julius Krumbiegel et al. Chain.jl [software], 2022. https://github.com/jkrumbiegel/Chain.jl.
[8] Harlan Harris; Tom Short; Chris DuBois; John Myles White; Milan Bouchet-Valat; Bogumił Kamiński; other DataFrames.jl contributors. DataFrames.jl [software], 2023. https://github.com/JuliaData/DataFrames.jl.
[9] Peter; Short, Tom; Deffebach et al. DataFramesMeta.jl [software], 2024.
[10] Karandeep Singh et al. TidierData.jl [software], 2023. https://github.com/TidierOrg/TidierData.jl.
[11] Hadley Wickham et al. dplyr: A grammar of data manipulation. 2023.

[5]`https://kezdigabor.life`