

Bringing data, automation and modelling together with Kotlin

A data-science journey from scripting to simulation

Dr. Holger Brandl
Analytics Solution Architect
2. November 2021

About Me



kscript

krangl

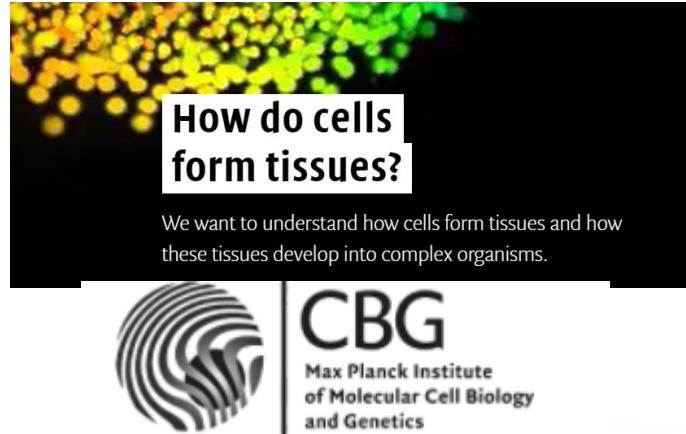
kravis

kalasim

2010

2015

2020



INDUSTRY 4.0



Watch the Kotlin 2021 Premier Online Event Keynote

Kotlin v1.5.31

Solutions Docs Community Teach Play

A modern programming language that makes developers happier.

Get started

Why Kotlin



Developed by [JetBrains](#) & Open-source [Contributors](#)



Multiplatform Mobile

The natural way to share code
between mobile platforms



Server-side

Modern development experience
with familiar JVM technology



Web Frontend

Extend your projects to web



Android

Recommended by Google for
building Android apps

February 2016: A new language was born

Kotlin v1.0 released

Type Inference

Extension Functions

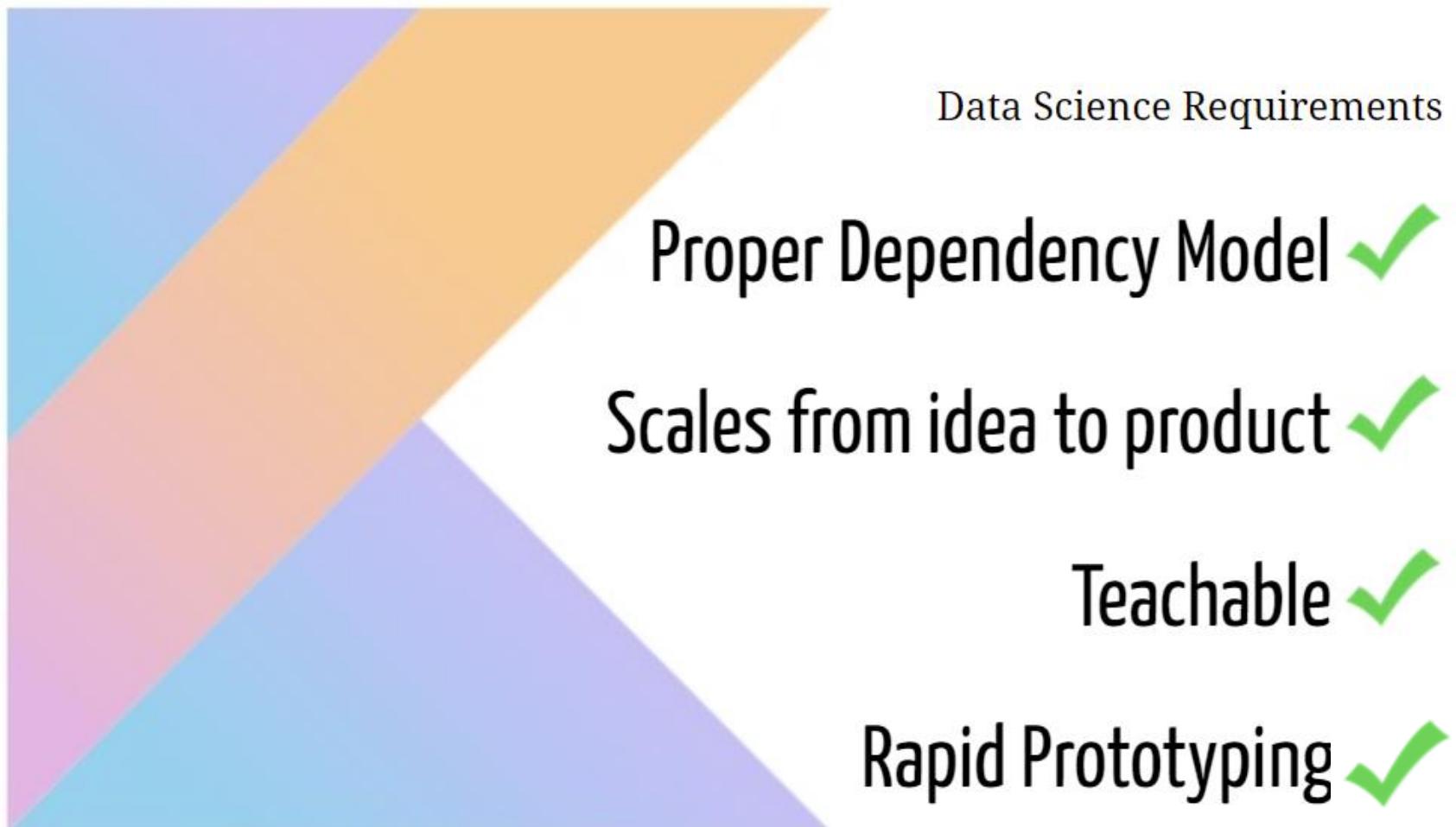
Data Classes

DSLs in Mind

Default Parameters

Lives in JVM

Scripting Support



Data Science Requirements

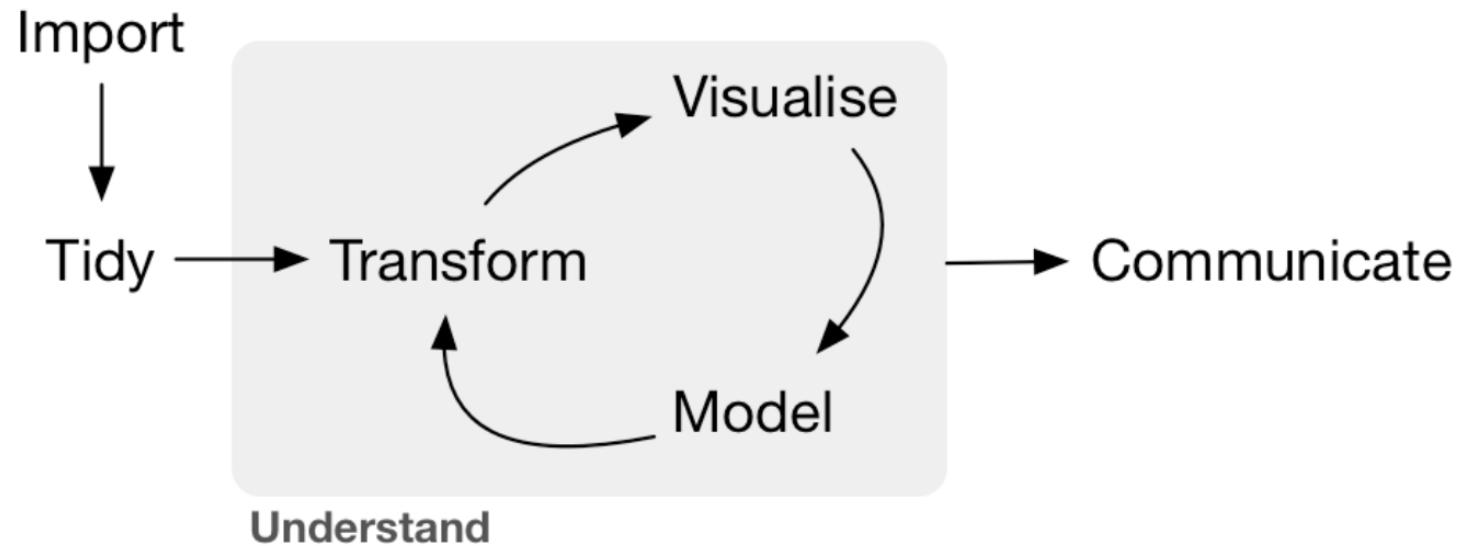
Proper Dependency Model ✓

Scales from idea to product ✓

Teachable ✓

Rapid Prototyping ✓

The data-science life cycle



- Python: pandas + scikit-learn + matplotlib/seaborn + jupyter
- R: readr + tidyverse + dplyr + ggplot2 + caret/broom + knitr
- JVM, and Kotlin in particular?



Search or jump to...

/ Pull requests Issues Marketplace Explore



holgerbrandl / kscript

Public

Unwatch 37

Unstar 1.6k

Fork

104

Code

Issues 29

Pull requests 4

Discussions

Actions

Projects

Wiki

Security

Insights

Settings

Enhanced scripting support for [Kotlin](#) on *nix-based systems.

Kotlin has some built-in support for scripting already but it is not yet feature-rich enough to be a viable alternative in the shell.

In particular this wrapper around kotlinc adds

- Compiled script caching (using md5 checksums)
- Dependency declarations using gradle-style resource locators More options to provide scripts including interpreter mode, reading from stdin, local files or URLs
- Embedded configuration for Kotlin runtime options
- Support library to ease the writing of Kotlin scriptlets
- Deploy scripts as stand-alone binaries

Use just the linux shell for data-driven science?

| But it's faster and way shorter to do it bash! (unknown colleague)

Do *magical* things in the terminal without dependencies

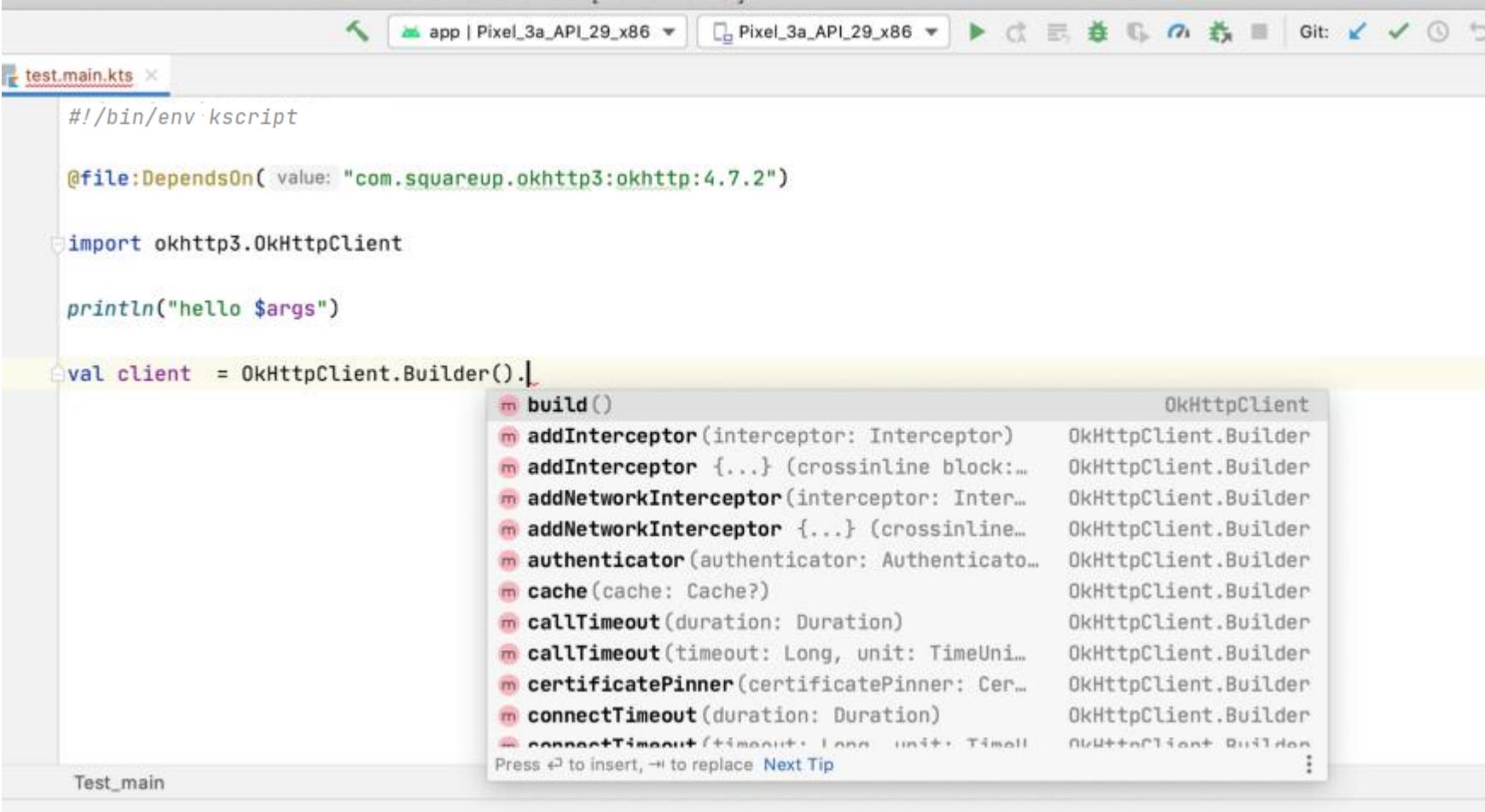
```
removeMultiMappers(){  
perl -ne 'unless((!($_=~/^@.*/) && !(m/AS:i:/)) || (m/AS:i:(\s+).+XS:i:(\s+)/ && $1-$2<1)){print;}'  
}  
  
bowtie2 $fastqFile | samtools view -SF 4 - | removeMultiMappers | samtools sort - result.bam
```

Marta Florio, Mareike Albert, Elena Taverna, Takashi Namba, **Holger Brandl**, Eric Lewitus, Christiane Haffner, Alex Sykes, Fong Kuan Wong, Jula Peters, E. Guhr, Sylvia Klemroth, Kay Prüfer, Janet Kelso, Ronald Naumann, Ina Nüsslein, Andreas Dahl, Robert Lachmann, Svante Pääbo, Wieland B. Huttner **Human-specific gene ARHGAP11B promotes basal progenitor amplification and neocortex expansion.** *Science* , 347(6229) 1465-1470 (2015)

Get published and forget.

Fast to write. Impossible to maintain or to evolve. Does not scale. Unteachable black art.

Scripting with kotlin: Fun, statically typed, with full IDE and library support



The screenshot shows an IDE interface with a tab bar at the top containing "app | Pixel_3a_API_29_x86" and "Pixel_3a_API_29_x86". Below the tabs is a toolbar with various icons. The main area displays a file named "test.main.kts". The code in the file is:

```
#!/bin/env kscript

@file:DependsOn( value: "com.squareup.okhttp3:okhttp:4.7.2" )

import okhttp3.OkHttpClient

println("hello $args")

val client = OkHttpClient.Builder().
```

A tooltip is displayed over the ".Builder()" call, listing several methods available on the OkHttpClient.Builder class:

- build()
- addInterceptor(interceptor: Interceptor) OkHttpClient.Builder
- addInterceptor {...} (crossinline block:...)
- addNetworkInterceptor(interceptor: Inter...) OkHttpClient.Builder
- addNetworkInterceptor {...} (crossinline...
- authenticator(authenticator: Authenticato...
- cache(cache: Cache?) OkHttpClient.Builder
- callTimeout(duration: Duration) OkHttpClient.Builder
- callTimeout(timeout: Long, unit: TimeUni...
- certificatePinner(certificatePinner: Cer...
- connectTimeout(duration: Duration) OkHttpClient.Builder
- connectTimeout(timeout: Long, unit: Tim...

At the bottom of the tooltip, it says "Press ⌘ to insert, ⌘ to replace Next Tip".

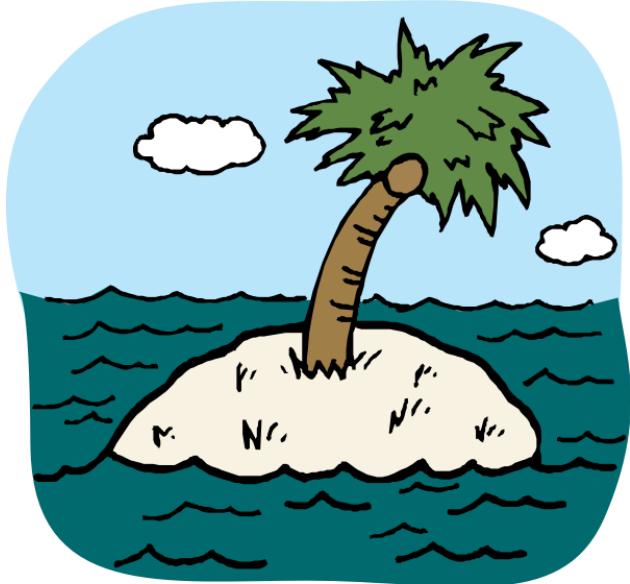
kscript Features Overview

- Installation
- Script Input Modes
- Script Configuration
- Text Processing Mode
- Treat yourself a REPL with `--interactive`
- Bootstrap IDEA from a `kscript let`
- Deploy scripts as standalone binaries
- Embed kscript installer within your script
- FAQ
- Support
- How to contribute?
- Acknowledgements

Further Reading

- <https://github.com/holgerbrandl/kscript>
- KotlinConf2017 : [kscript - Scripting Enhancements for Kotlin](#)
- [Talking Kotlin](#) with Holger Brandl about kscript

What library would I take to a lonely island?



Search...

Get started

Reference

Articles ▾

News ▾



Overview

dplyr is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.
- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

These all combine naturally with `group_by()` which allows you to perform any operation “by group”. You can learn more about them in `vignette("dplyr")`. As well as these single-table verbs, dplyr also provides a variety of two-table verbs, which you can learn about in `vignette("two-table")`.

If you are new to dplyr, the best place to start is the [data transformation chapter](#) in R for data science.

Links

Download from CRAN at
[https://cloud.r-project.org/
package=dplyr](https://cloud.r-project.org/package=dplyr)

Browse source code at
<https://github.com/tidyverse/dplyr/>

Report a bug at
[https://github.com/tidyverse/dplyr/
issues](https://github.com/tidyverse/dplyr/issues)



Check what else is out there

- [tablesaw](#) which is (according to its authors) the *The simplest way to slice data in Java*
- [kotlquery](#) is a handy database access library
- [Scala DataTable](#): a lightweight, in-memory table structure written in Scala
- [joinery](#) implements data frames for Java
- [paleo](#) which provides immutable Java 8 data frames with typed columns
- [morpheus-core](#) which is a data science framework implementing an R-like data-frame
- [vectorz](#) is a fast and flexible numerical library for Java featuring N-dimensional arrays
- [koma](#) is a scientific computing library written in Kotlin
- [termsql](#) converts text from a file or from stdin into SQL table using sqlite and query it instantly

Great stuff, but not feature-complete enough, maintained or *kotlinesque* as needed for fluency and fun with data



Search or jump to...

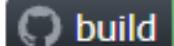
Pull requests Issues Marketplace Explore



holgerbrandl / krangl Public

[Unwatch](#) 11[Unstar](#) 476[Fork](#) 50[Code](#)[Issues 19](#)[Pull requests 1](#)[Discussions](#)[Actions](#)[Projects](#)[Wiki](#)[Security](#)[Insights](#)[Settings](#)

Maven Central 0.17



build passing

[chat](#)[on gitter](#)

`krangl` is a {K}otlin library for data wrangling. By implementing a grammar of data manipulation using a modern functional-style API, it allows to filter, transform, aggregate and reshape tabular data.

`krangl` is heavily inspired by the amazing `dplyr` for R. `krangl` is written in Kotlin, excels in Kotlin, but emphasizes as well on good java-interop. It is mimicking the API of `dplyr`, while carefully adding more typed constructs where possible.

<https://github.com/holgerbrandl/krangl>

Copy Learn from the best, learn from dplyr

Data Transformation with dplyr :: CHEAT SHEET

dplyr functions work with pipes and expect `tidy data`. In tidy data:

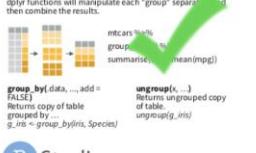


Summarise Cases

These apply `summary` functions to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value as output.

Group Cases

Use `group_by` to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately, then combine the results.



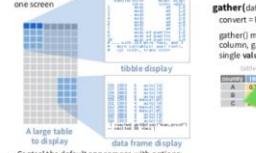
Tibbles

An enhanced data frame. The `tibble` package provides a new S3 class for working tabular data, the `tibble` class. It's similar to the `data.frame` class, but improves three behaviors:

- Subsetting - always return a new tibble, `[::]` & `[[::]]` always return a vector.

- No partial matching - You must use full column names when substringing.

- Display - When you print a tibble, R provides a concise view of the data that fits on one screen.



- Control default performance with options: `options(tibble.print_min=10, tibble.print_max=10)`
- View full data set with `View()` or `glimpses()`
- Revert to data frame with `as.data.frame()`

CONSTRUCT A TIBBLE IN TWO WAYS

`tibble()`
- Construct by columns:
`tibble(x=1, y=2, z=3)`

`as_tibble()`
- Convert data frame to tibble
`enframe(name = "name", value = "value")`

Manipulate Cases

EXTRACT CASES

`Row` functions return a subset of rows as a new table.

`filter`, `data[,]` Extract rows that meet logical criteria. `filter(iris, Sepal.Length > 7)`

`distinct`, `keep = FALSE`) Remove rows with duplicate values.

`sample`, `frac(1, size = 1)`, `replace = FALSE`) Compute random sample of rows. `iris %>% sample(1, size = 1, replace = FALSE)`

`sample_n`, `size`, `replace = TRUE`) Compute random sample of rows. `iris %>% sample_n(10, replace = TRUE)`

`slice`, `n = 10`) Select and order top n entries (by group if grouped data). `iris %>% slice(10, 1:5)`

`top_n`, `x, wt`) Select and order top n entries (by group if grouped data). `iris %>% top_n(5, Sepal.Width)`

`VARIATIONS`

`summarise_all`) Apply fun to every column.

`summarise_if`) Apply fun to specific columns.

`summarise_at`) Apply fun to all cols of one type.

Group Cases

Use `group_by` to create a "grouped" copy of a table.

dplyr functions will manipulate each "group" separately, then combine the results.



Manipulate Variables

EXTRACT VARIABLES

`Column` functions return a set of columns as a new vector or table.

`pull`, `data, var = -1`) Extract column values as a vector. Choose by name or index. `pull(iris, Sepal.Length)`

`select`, `ct, ...`) Extract columns as a table. Also `select_if`, `select_where`.

`sample_n`, `size`, `replace = TRUE`) Compute random sample of rows. `iris %>% sample_n(10, replace = TRUE)`

`distinct`, `match`, `ends`, `with_in`) One or more columns starts_with(match)

`matches`(`match`)

`starts_with`(`match`)

`MAKE NEW VARIABLES`

These apply `vectorized functions` to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

`vectorized function`

`mutate`, `data, ...`) Compute new column, drop others. `iris %>% mutate(Sepal.Length, gmp = 1/mprg)`

`transmute`, `data, ...`) Compute new columns, drop others. `iris %>% transmute(Sepal.Length, gmp = 1/mprg)`

`mutate_all`, `funs, ...`) Apply funs to every column. `iris %>% mutate_all(funs, log10)`

`mutate_if`, `tbl, funs, ...`) Apply funs to specific columns. `iris %>% mutate_if(Sepal.Length, log10)`

`mutate_at`, `tbl, cols, funs, ...`) Apply funs to element across a set of vectors. `iris %>% mutate_at(Sepal.Length:Sepal.Width, log10)`

`mutate`, `at`, `vars, funs, ...`) Replace specific values with NA. `iris %>% mutate(at(Sepal.Length), max = 1)`

`mutate`, `at`, `vars, fns, ...`) Compute mean for point numbers

`MISC`

`case_when`) multi-case if else

`case_fold`) first non-NA values by element across a set of vectors.

`copy_to`, `tbl, dest = "dest"`) else

`copy_na`, `tbl`) replace specific values with NA

`max`) element-wise max

`min`) element-wise min

`recode`) Vectorized switch

`recode_factor`) Vectorized switch for factors



Vector Functions

TO USE WITH MUTATE()

`mutate()` (and `mutem`) apply vectorized functions to columns to create a new table. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function

`OFFSETS`

`dplyr::lag()` Offset elements by 1

`dplyr::lead()` Offset elements by -1

CUMULATIVE AGGREGATES

`dplyr::cumall()` Cumulative all

`dplyr::cumany()` Cumulative any

`dplyr::cummean()` Cumulative mean

`dplyr::cumprod()` Cumulative prod

`dplyr::cumsum()` Cumulative sum

LOCATION

`mean`) -mean, `median`) median

LOGICALS

`mean`) - proportion of TRUE's

`sum`) - or TRUE's

RANKING

`dplyr::cume_dist()` Proportion of all values =<

`dplyr::dense_rank()` rank with ties = min, no gaps

`dplyr::rank()` rank with ties = min

`dplyr::percent_rank()` min, rank scaled to [0,1]

`dplyr::row_number()` rank with ties = "first"

MATH

`*`, `^`, `%,`, `%%`, `-`, `/`, `-`, `+`) arithmetic ops

`log10`, `log10`) log base 10

`log`, `log`) log base e

`dplyr::between()` - x >= left & x <= right

`dplyr::near()` - near for floating point numbers

SPIECE

`iqr`) - inter-quartile range

`median`) median absolute deviation

`sd`) standard deviation

`var`) variance

MISC

`case_when`) - multi-case if else

`case_fold`) first non-NA values by element across a set of vectors.

`copy_to`, `tbl, dest = "dest"`) else

`copy_na`, `tbl`) replace specific values with NA

`max`) element-wise max

`min`) element-wise min

`recode`) Vectorized switch

`recode_factor`) Vectorized switch for factors

Row Names

Tidy data stores row names, which atomizes the variable outside of the columns. To work with the rownames, first move them into a column.

`rownames_to_column`) Move row names to column.

`column_to_rownames`) Move col names to row names.

`rownames`) Set row names to col.

`row.names`) Set row names to col.

`row.names = "row.names"`) Set row names to col.

How to get your data into krangl

Define tables inline

```
val users = DataFrameOf(  
    "firstName", "lastName", "age", "hasSudo")(  
    "max", "smith" , 53, false,  
    "eva", "miller", 23, true,  
    null , "meyer" , 23, null)
```

Read & write from tsv, csv, json, jdbc, etc.

```
val tornados = DataFrame.readCSV(pathAsStringFileOrUrl)  
// do stuff  
tornados.writeCSV(File("tornados.txt.gz"))
```

- Guess column types & default parameters
- Built-in missing value support

Example: Summarize Flights by Date

```
flightsData
    .groupBy("year", "month", "day")
    .select({ range("year", "day") }, { listOf("arr_delay", "dep_delay") })
    .summarize(
        "mean_arr_delay" `= ` { it["arr_delay"].mean(removeNA = true) },
        "mean_dep_delay" to { it["dep_delay"].mean(removeNA = true) }
    )
    .filter { (it["mean_arr_delay"] gt 30) OR (it["mean_dep_delay"] gt 30) }
    .sortedBy("mean_arr_delay")
```

year	month	day	mean_arr_delay	mean_dep_delay
2013	10	11	18.9229	31.2318
2013	5	24	24.2574	30.3407
2013	6	2	26.075	34.0133
2013	6	26	27.3174	30.61175
2013	6	10	28.0222	30.61945
2013	7	8	29.6488	37.2966
2013	8	22	29.9767	33.6004
2013	2	27	31.252	37.7632

To type or not to type?

- *Static types* are cool, but most data has no type
- It's more robust/fun to use types and they allow for better design
- Many data attributes are very fluent

```
data class Employee(val id:Int, val name:String)
val staffStats = listOf(Employee(1, "John"), Employee(2, "Anna"))
    .predictNumSickDays()      // new type!
    .addPerformanceMetrics()   // new type!
    .addSalaries()            // new type!
    .correlationAnalysis()    // odd generic signature :-/
```

- R/python lack static typing, which make such workflows more fluent/fun to write

```
staff %>%
  mutate(sick_days=predictSickDays(name)) %>%      # table with another column
  left_join(userPerf) %>%                          # and some more columns
  left_join(salaries) %>%                          # and even more columns
  select_if(is.numeric) %>%
  correlate(type="spearman")                         # correlate numeric attributes
```

Mix typed and untyped data

```
val dataFrame : DataFrame =
```

employee:Employee	sales>List<Sale>	age:Int	address:String	salary:Double
Employee(23, "Max")	listOf(Sale(...), Sale())	23	"Frankfurt"	50.3E3
...

```
// aggregations like
dataFrame.groupBy("age").count()
dataFrame.summarize("mean_salary"){ mean(it["salaray"])}

// integration like
val df: DataFrame = dataFrame.leftJoin(otherDF)

// transformations like
dataFrame.addColumn("intial"){ it["employee"] .map<Employee>{ it.name.first() }}
```

- krang! includes methods to go back and forth between untyped and typed data
- Best method ever `asDataFrame()` for every Iterable
- Flattening of complex types with `df.unfold<Car>("vehicle")`

Bumpy API corners in krang!

Lists in table formulas cause operator confusion

```
users.addColumn("age_plus_3") { it["user"].map<User> { it.age } + 3 } // extend list with 3
users.addColumn("age_plus_3") { it["user"].map<User> { it.age + 3 } } // correct
users.addColumn("age_plus_3") { it["age"] + 3 } // works because `DataCol.plus` can be defined
```

Incomplete vectorization for operators

Some (+, -, *, !) can be overridden for collections, but others cannot (e.g. all arithmetic and boolean comparison ops)

No vectorization for >, && ==, etc. in table formulas → Use function calls or not so pretty gt, AND, eq, etc.

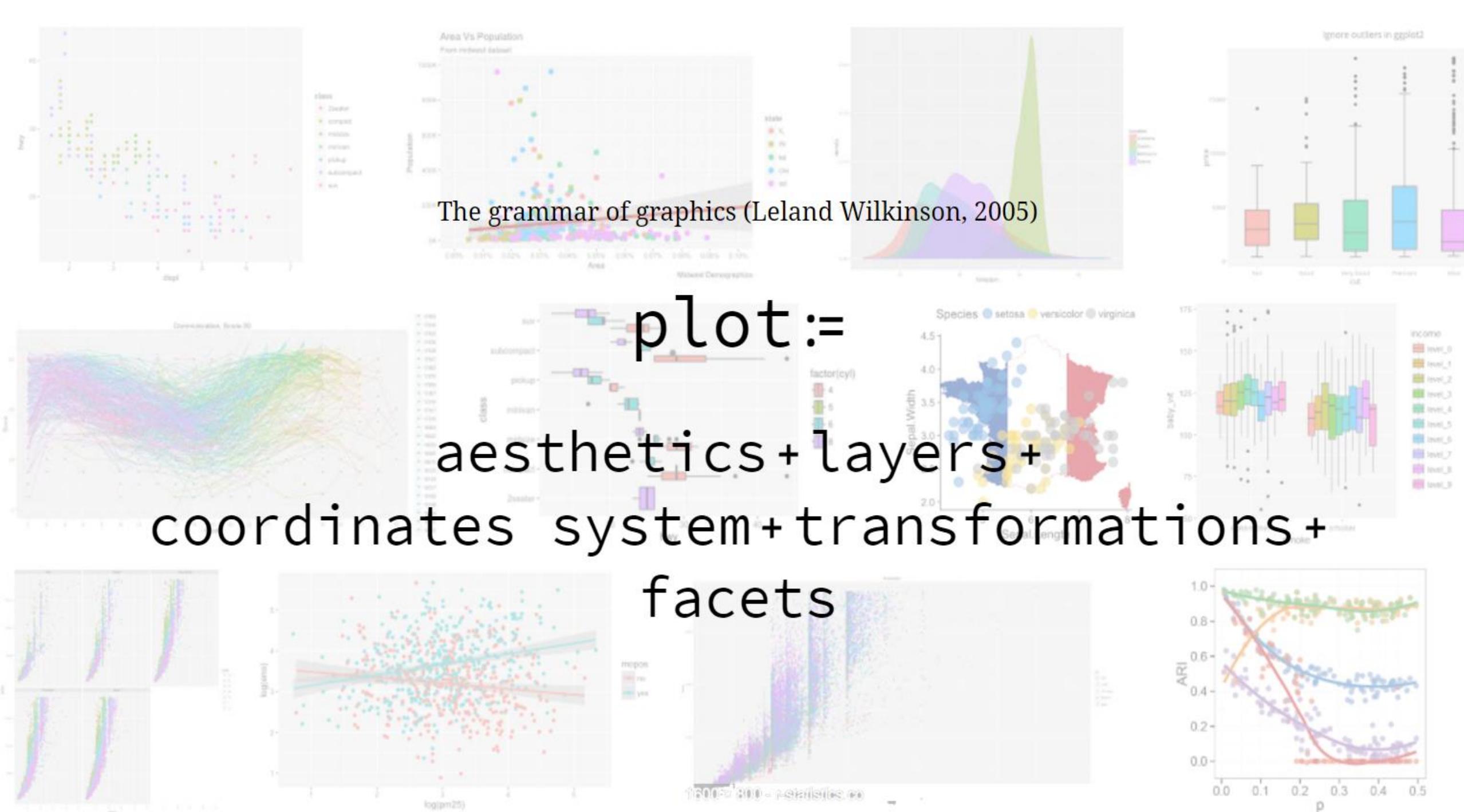
Next steps

Promising API, great learning experience.

Project still in flux towards a convenient and consistent API design

Next steps

- Date column support
- Better lambda receiver contexts
- Performance (indices, avoid list and array copies, compressed columns)
- Use dedicated return type for table formula helpers (like `mean`, `rank`) to reduce runtime errors
- More bindings to other jvm data-science libraries
- Better documentation & cheatsheets
- Sequence vs `Iterable`?
- Pluggable backends like native or SQL





Search or jump to...

Pull requests Issues Marketplace Explore



holgerbrandl / kravis Public

[Unwatch](#) 1[Unstar](#) 145[Fork](#) 8[Code](#) [Issues 7](#) [Pull requests 1](#) [Discussions](#) [Actions](#) [Projects](#) [Wiki](#) [Security](#) [Insights](#) [Settings](#)

kravis - A {k}otlin {gra}mmar for data {vis}ualization

[Maven Central](#) 0.8.1 [build](#) failing

Visualizing tabular and relational data is the core of data-science. `kravis` implements a grammar to create a wide range of plots using a standardized set of verbs.

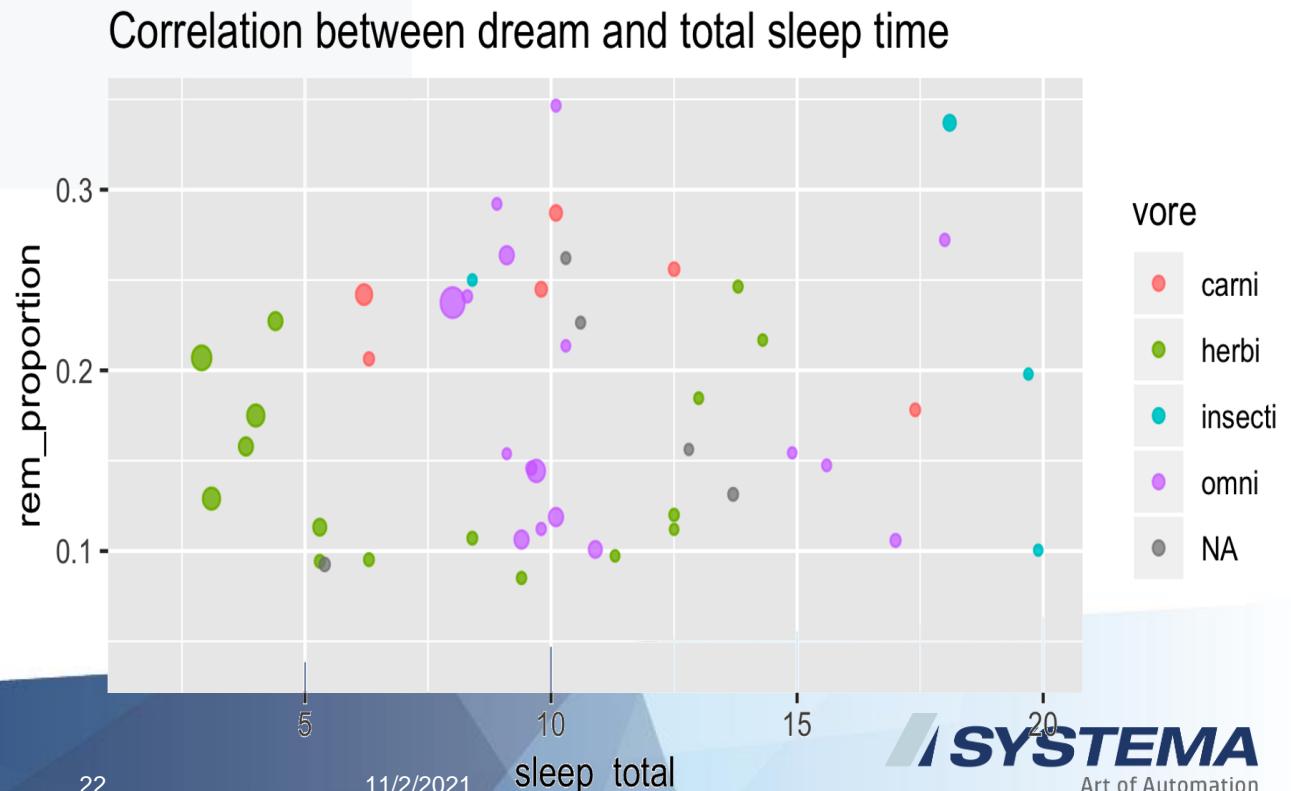
The grammar implemented by `kravis` is inspired from `ggplot2`. In fact, all it provides is a more typesafe wrapper around it. Internally, `ggplot2` is used as rendering engine. The API of `kravis` is highly similar to allow even reusing their excellent [cheatsheet](#).

R is required to use `ggplot`. However, `kravis` works with various integration backend ranging such as docker or remote webservices.

<https://github.com/holgerbrandl/kravis>

Can we map a scripting API into a statically typed language?

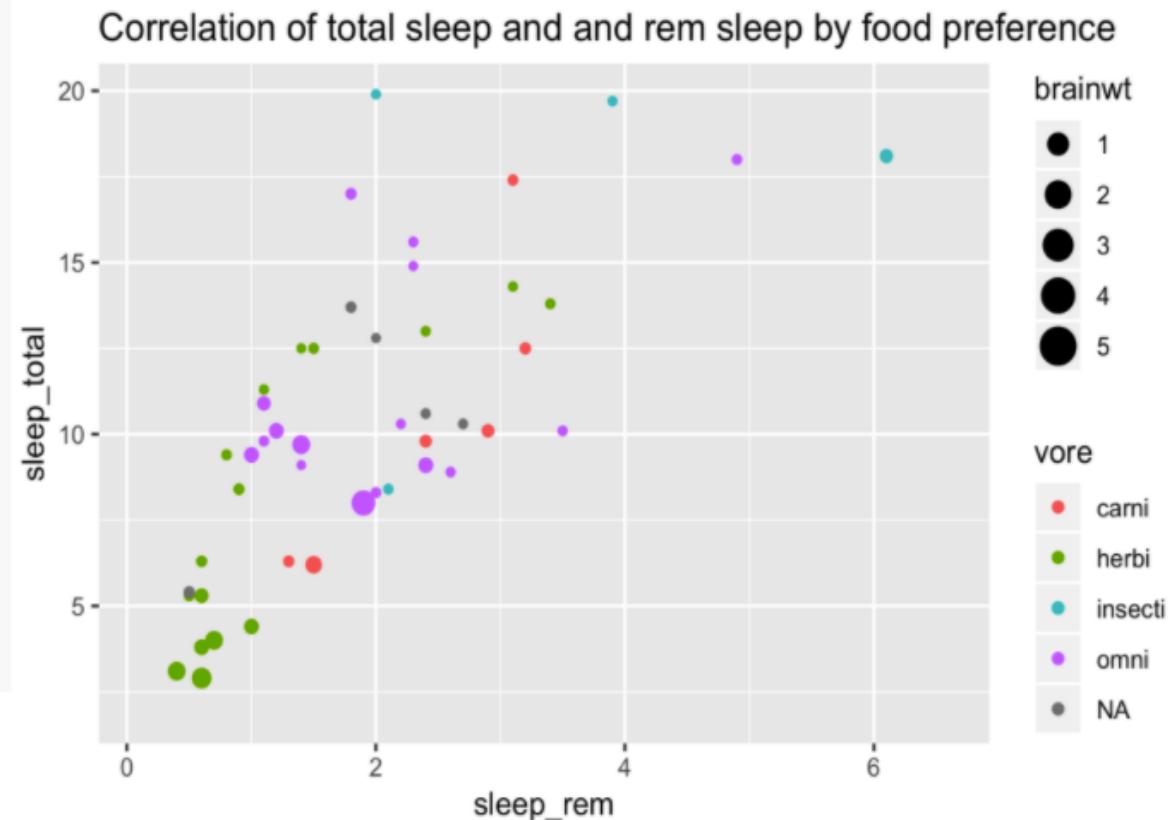
```
import krangl.*  
import kravis.*  
  
sleepData  
    .addColumn("rem_proportion") { it["sleep_rem"] / it["sleep_total"] }  
    // Analyze correlation  
    .plot(x = "sleep_total", y = "rem_proportion", color = "vore", size = "brainwt")  
        .geomPoint(alpha = 0.7)  
        .guides(size = LegendType.none)  
    .title("Correlation between dream and total sleep time")
```



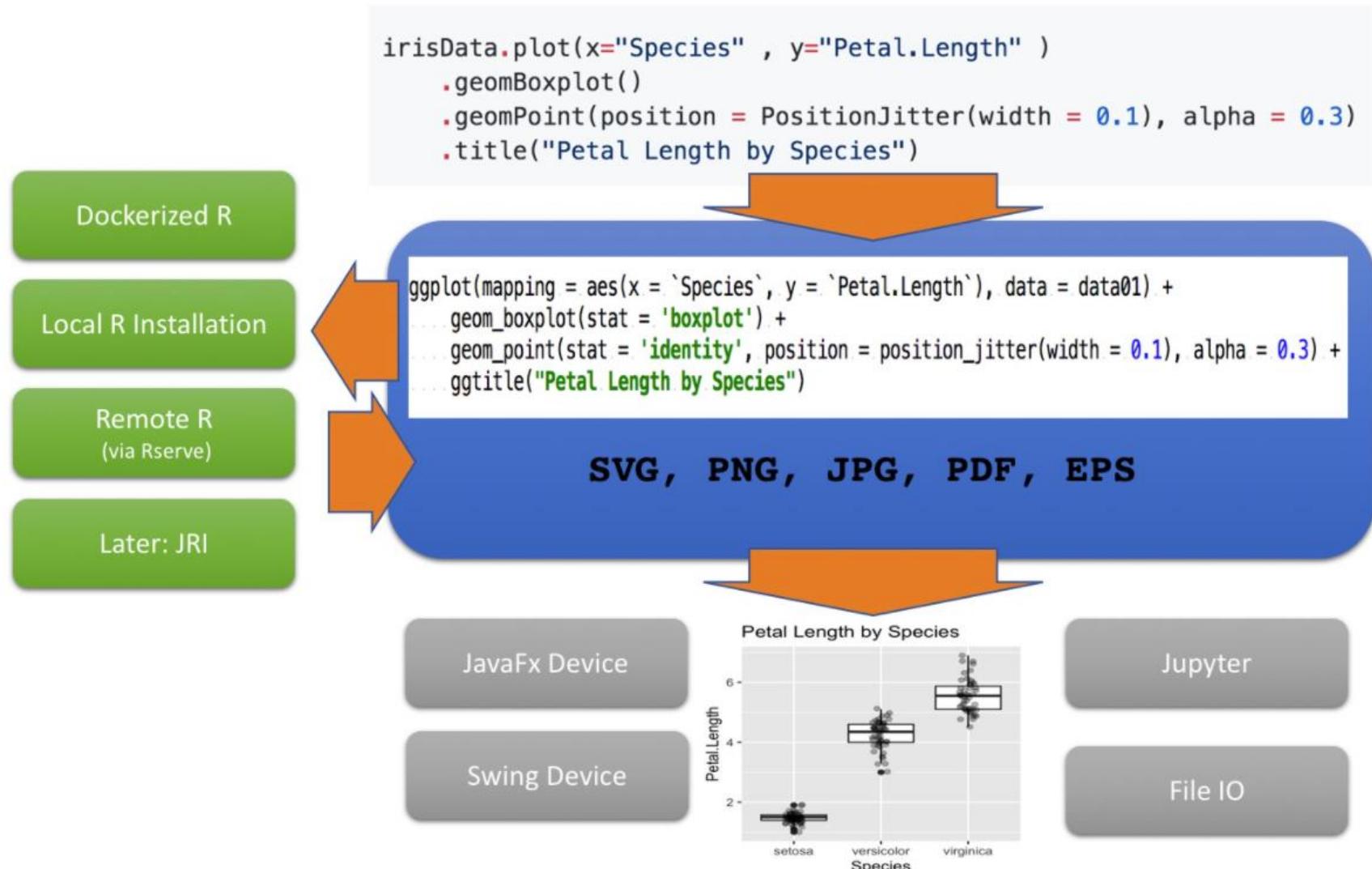
Example: Typed API

Use any `Iterable<T>` to build a plot. :-)

```
val sleepPatterns : List<SleepPattern> = ...  
  
// deparse records using property references  
sleepPatterns  
    .plot(  
        x = SleepPattern::sleep_rem,  
        y = SleepPattern::sleep_total,  
        color = SleepPattern::vore,  
        size = SleepPattern::brainwt  
    )  
    .geomPoint()  
    .title("Correlation of total sleep " +  
        "and rem sleep by food preference")
```



System Architecture



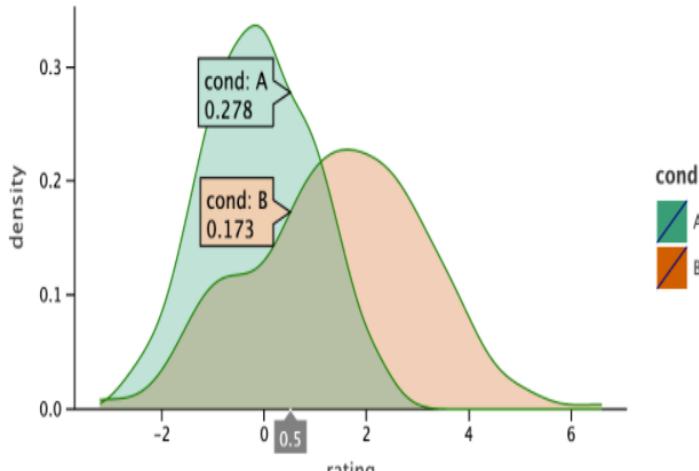
lets-plot as modern alternative

- Removes annoying dependency on R (which is great!!)
- lets-plot provides a kotlin-native rendering backend, more interactivity
- Currently, still limited method support and customizability compared to kravis

```
val rand = java.util.Random()
val data = mapOf<String, Any>(
    "rating" to List(200) { rand.nextGaussian() } + List(200) { rand.nextGaussian() * 1.5 + 1.5 },
    "cond" to List(200) { "A" } + List(200) { "B" }
)

var p = lets_plot(data)
p += geom_density(color="dark_green", alpha=.3) {x="rating"; fill="cond"}
p + ggsize(500, 250)
```

- Execute the added code to evaluate the plotting capabilities of Lets-Plot.



Data Science with Kotlin Tutorial

→ Mammalian Sleep Patterns



Further reading: Data science with kotlin

Tutorials

- <https://kotlinlang.org/docs/data-science-overview.html>
- http://holgerbrandl.github.io/krangl/10_minutes/
- http://holgerbrandl.github.io/krangl/data_manip/

Talks

- KotlinConf2017 - [A wild ride through New York City](#) with Holger Brandl
- KotlinNight2018 - [Kotlin's emerging data-science ecosystem](#) with Holger Brandl
- KotlinConf 2019 [Making Kotlin Ready for Data Science](#) by Roman Belov

Social

- Kotlin Slack channels [science](#) and [data-science](#) & github



Digital Transformation

Smart Manufacturing

SYSTEMA Portfolio

SAP Portfolio

Resources

Integration

Automation

Optimization

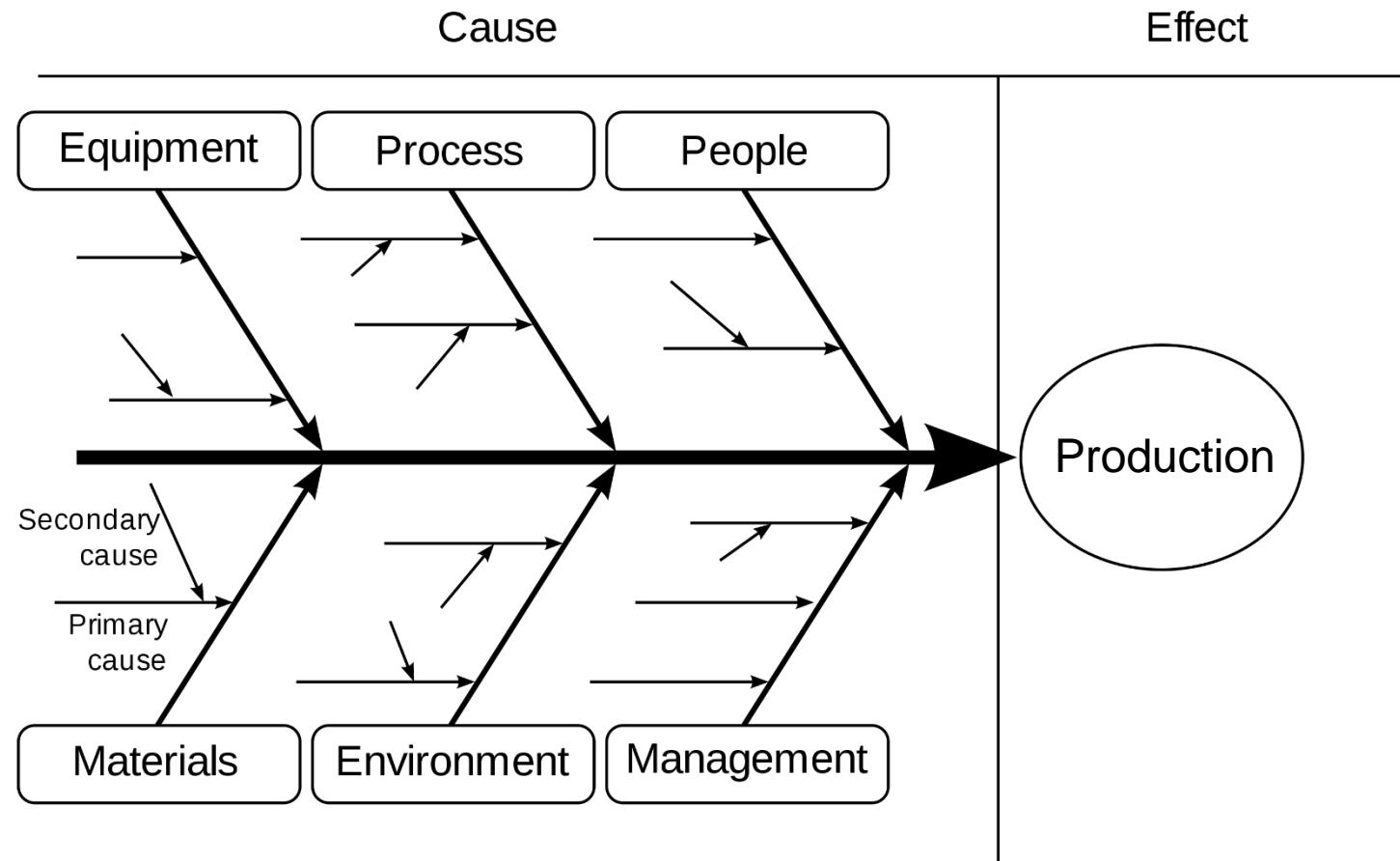
Visualization

Migration



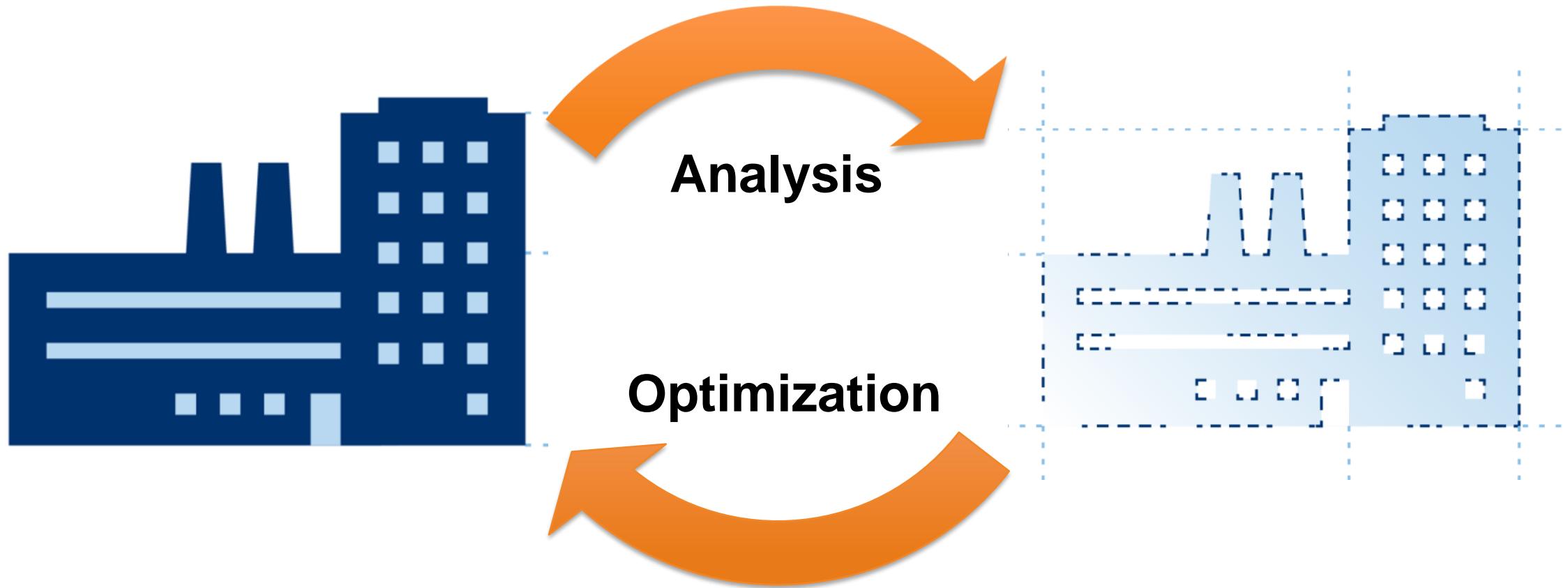
Optimization is the art of maximizing manufacturing efficiency, throughput, OEE, yield, and quality by monitoring, analyzing, and iteratively tuning manufacturing processes.

Cause and consequence in production planning

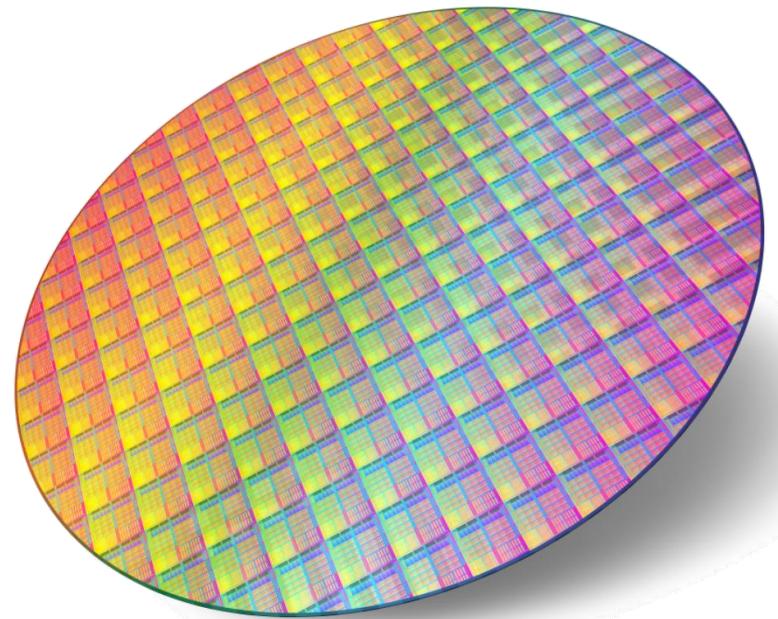
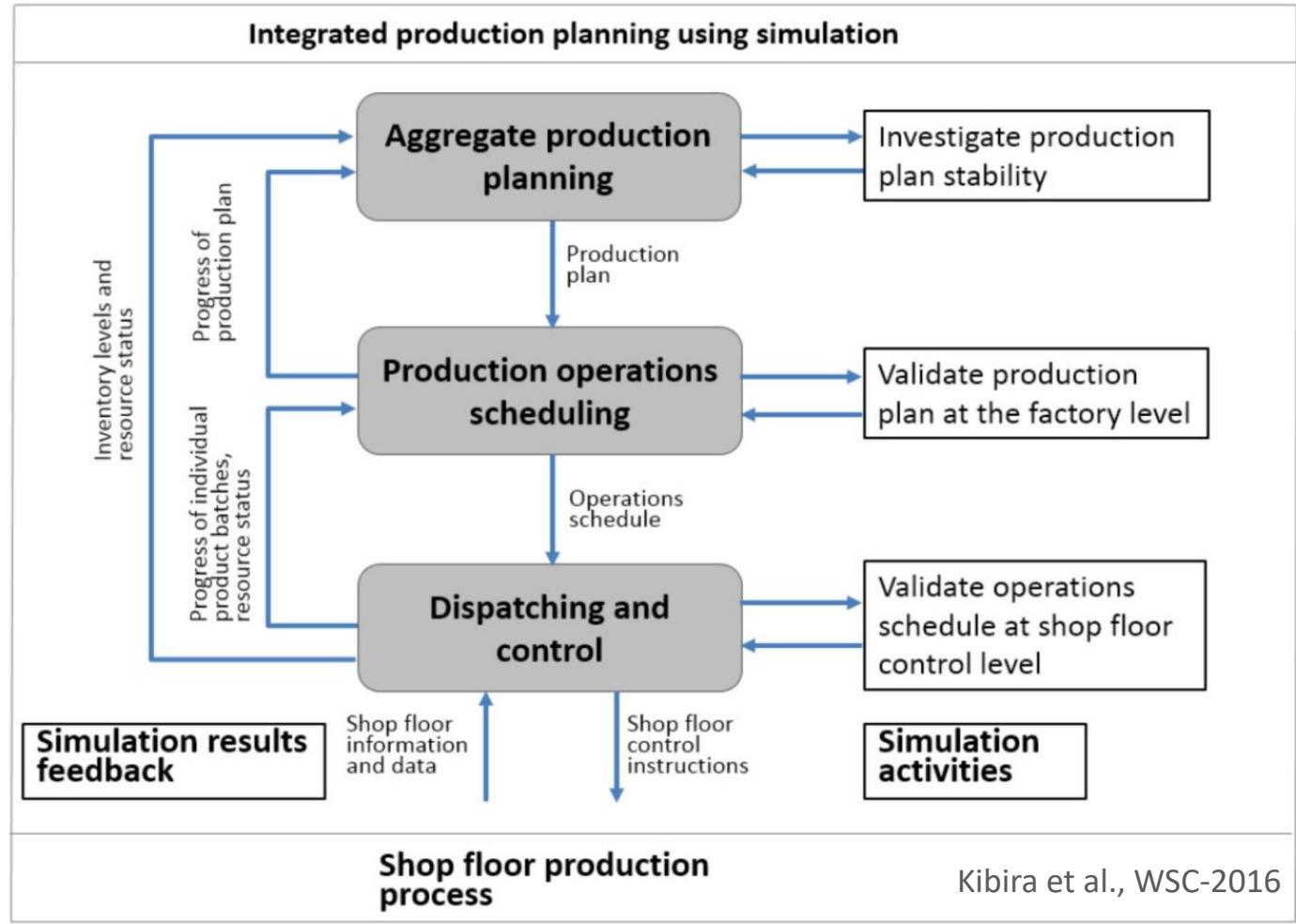


- Complex interplay of human, machine, material and methods
- Not always following intuition

How to optimize a complex system?



How to produce more in complex industries?





Welcome to kalasim

[Table of contents](#)[Core Features](#)[First Example](#)[How to contribute?](#)[Support](#)[release v0.6.90](#) [build](#) [passing](#) [gitter](#) [join chat](#) [kotlinlang slack](#) [kalasim](#)

`kalasim` is a [discrete event simulator](#) that enables complex, dynamic process models. It provides a statically typed API, dependency injection, modern persistence, structured logging and automation capabilities.

`kalasim` is written in [Kotlin](#), is designed around suspendable [coroutines](#) for process definitions, runs on the [JVM](#) for performance and scale, is built with [koin](#) as dependency wiring framework, and is using [common-math](#) for stats and distributions. See [acknowledgements](#) for further references.

Core Features

`kalasim` is a generic **process-oriented** discrete event simulation (DES) engine.

- **Simulation entities** have a generative process description that defines the interplay with other entities
- There is a well-defined rich process interaction vocabulary, including **hold**, **request**, **wait** or **passivate**
- An **event trigger queue** maintains future action triggers and acts as sole driver to progress simulation state
- Built-in **monitoring** and **statistics** gathering across the entire API

First Example

Key Types

- Component
- Resource
- State

Key methods

- request
- hold

```
////Cars.ks
import org.kalasim.*

class Driver : Resource()
class TrafficLight : State<String>("red")

class Car : Component() {

    val trafficLight = get<TrafficLight>()
    val driver = get<Driver>()

    override fun process() = sequence {
        request(driver) {
            hold(1.0, description = "driving")

            wait(trafficLight, "green")
        }
    }
}

createSimulation(enableConsoleLogger = true) {
    dependency { TrafficLight() }
    dependency { Driver() }

    Car()
}.run(5.0)
```

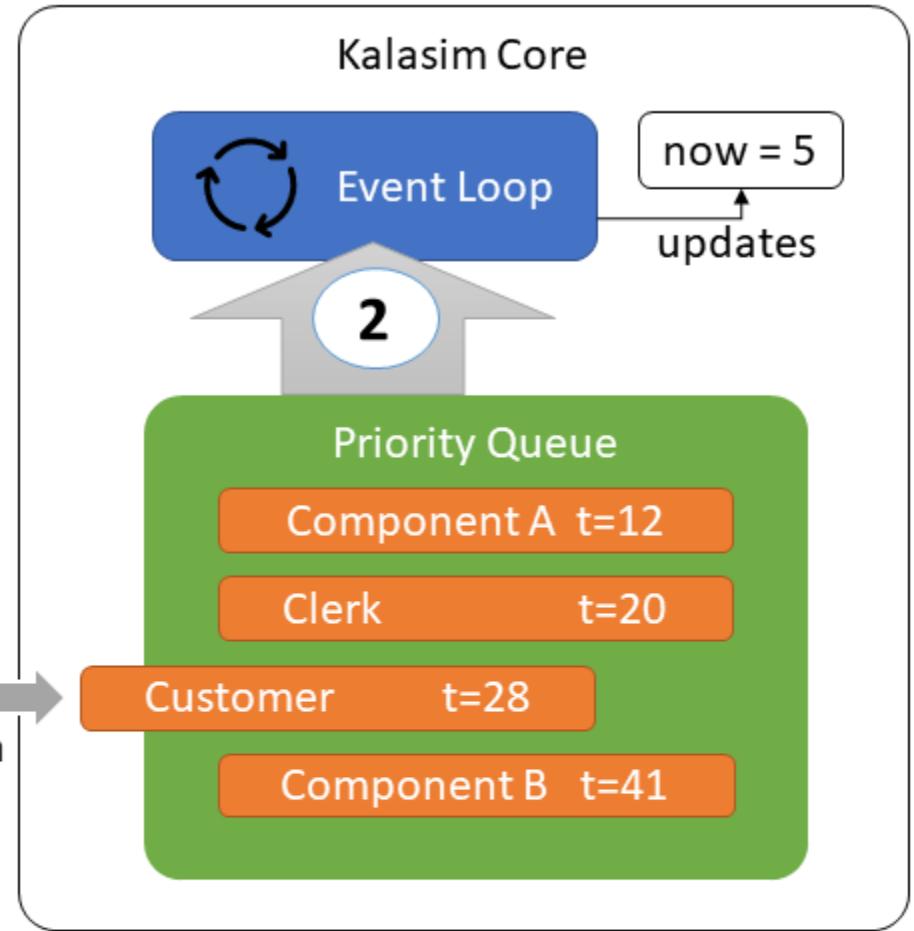
Event Loop Architecture

```
class Customer(val clerk: Resource) : Component() {  
  
    override fun process(): Sequence<Component> = sequence {  
  
        // do shopping  
        hold(ticks = 23.0)  
  
        // wait for an empty counter  
        request(clerk)  
  
        // billing process  
        hold(ticks = 2.0, priority = HIGH)  
    }  
}
```

3

1

now + duration



1 Stall customer's process execution and insert it into event queue by time (and optional priority to resolve ambiguities)

2 Poll, while queue is not empty and resume (or start) process

3 Event loop will continue execution here. Components are terminated once process end has been reached

Why another DES engine?

- Simmer, Salabim, SimJulia, SimPy, DSOL, ...
- UI-driven commercial vendors: AnyLogic, SIMIO

Kotlin vs Python

Static vs Dynamic

Python is a powerful, flexible platform with a simple syntax and rich ecosystem of libraries.

Dynamic typing makes Python flexible for ad hoc analysis, but it is challenging to use in production.

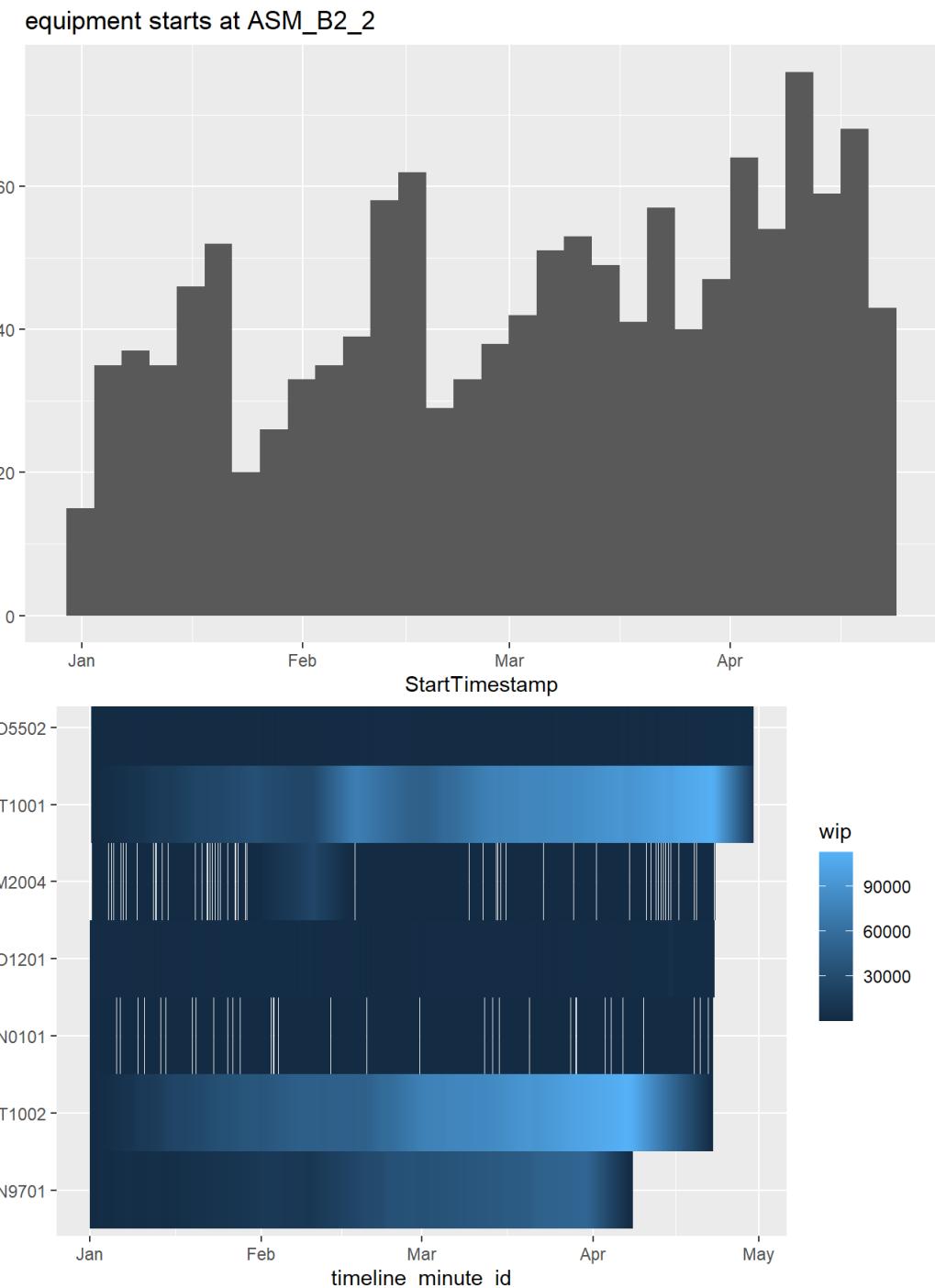
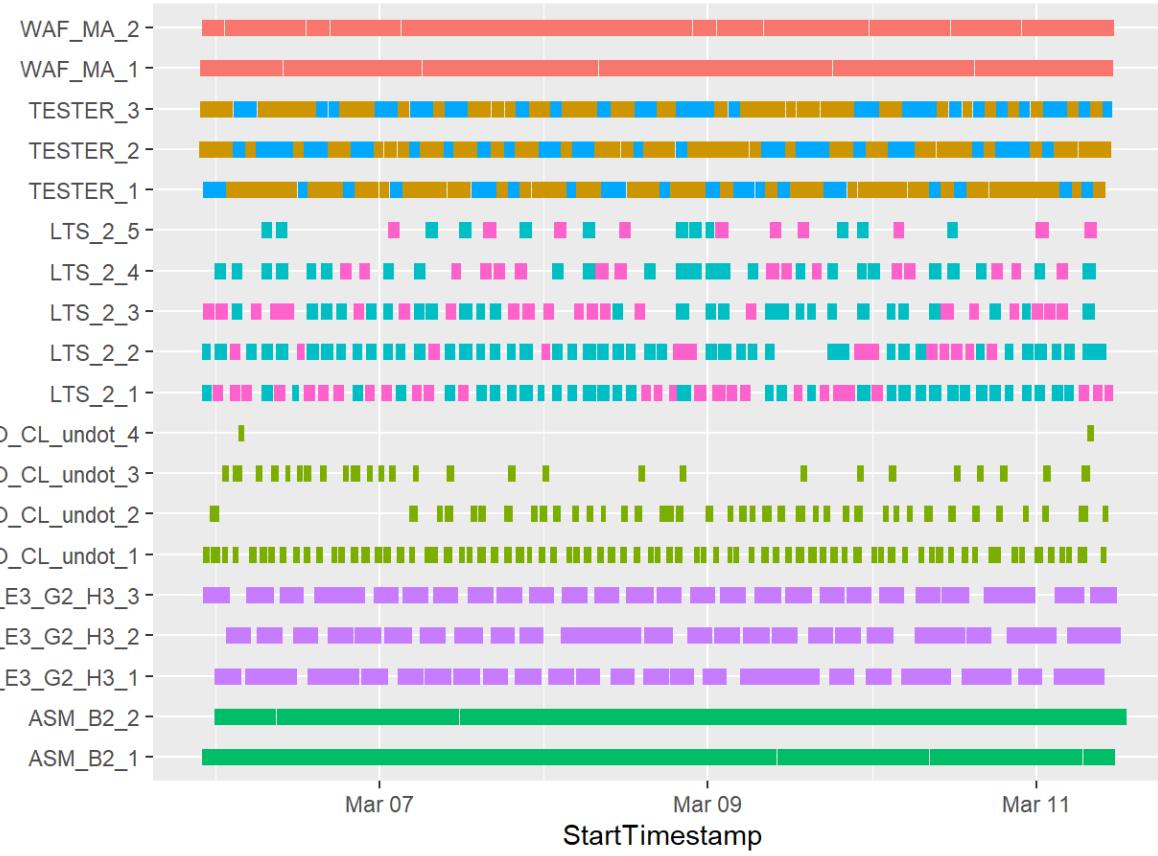
- Dynamic types allow improvised data structures to be defined at runtime.
- Dynamic typing can quickly create difficulties in maintaining, testing, and debugging codebases, especially as the codebase grows large.

One language, One Codebase, One Platform

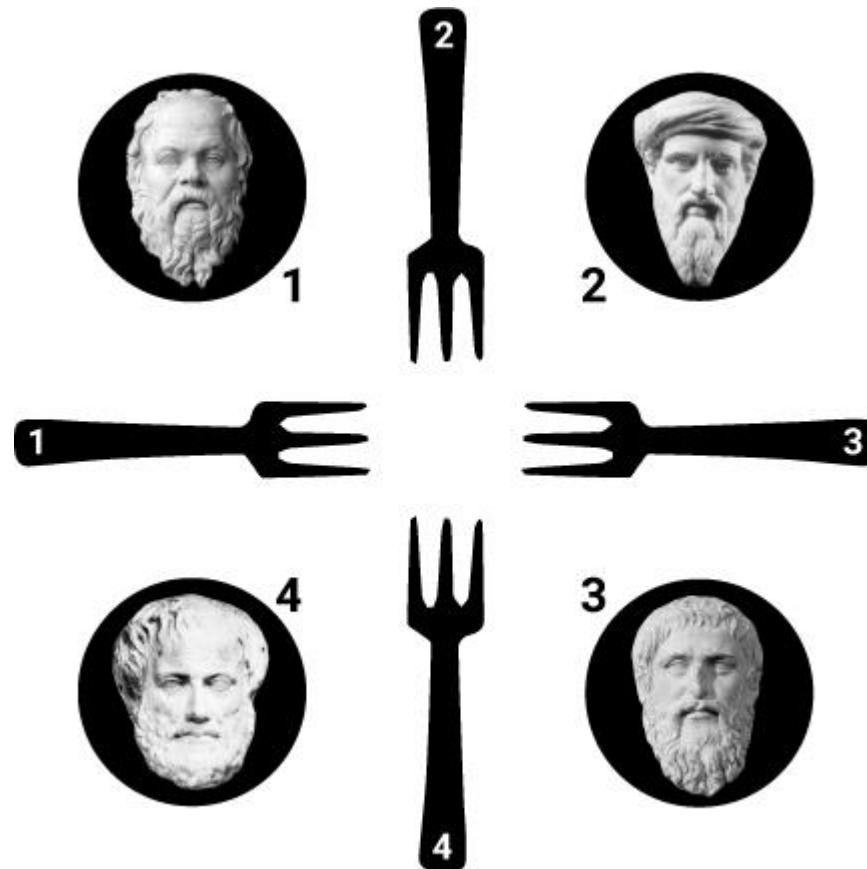


Can we model a semiconductor fab with kalasim?

- Yes, we can!



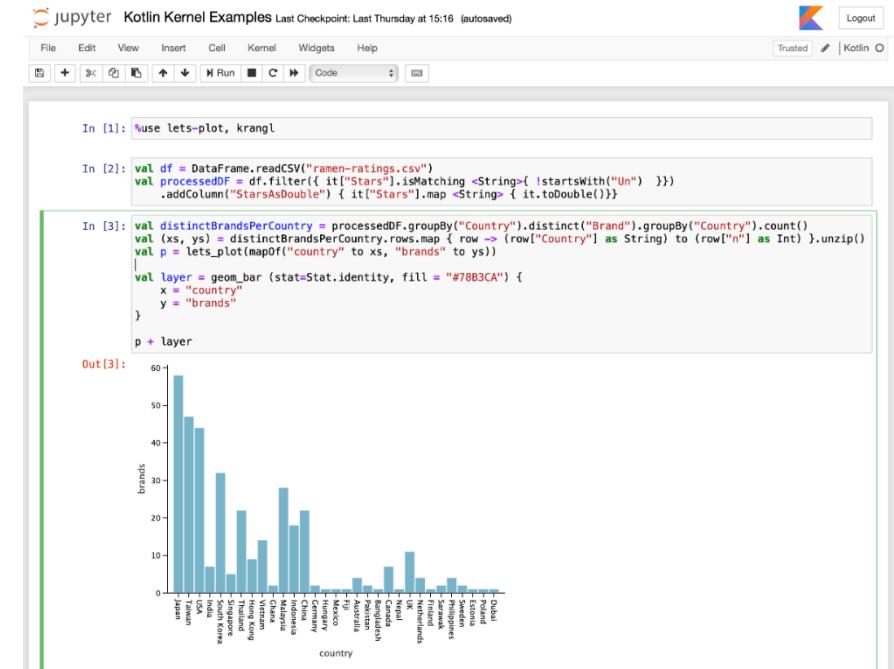
Example: Let's have some food with Dining Philosophers



https://www.kalasim.org/examples/dining_philosophers/

Quo vadis, kotlin for science?

- Amazing language for modelling & prototyping
- Great projects [KotlinDL](#) or [multik](#)
- Awesome [jupyter kernel](#)
- Tooling barrier easy to climb for developers but harder for data scientists
- Slow data-science community adoption
- Multiplatform & Huge JVM ecosystem



The screenshot shows a Jupyter Notebook interface titled "Jupyter Kotlin Kernel Examples". The notebook has a "Trusted" status and a "Logout" button. The code in cell [3] is as follows:

```
In [1]: use lets_plot, krangl
In [2]: val df = DataFrame.readCSV("ramen-ratings.csv")
val processedDF = df.filter{ it["Stars"] .isMatching <String>{ it.startsWith("Un" ) } }
.addColumn("StarsAsDouble") { it["Stars"].map<String>{ it.toDouble() }}
In [3]: val distinctBrandsPerCountry = processedDF.groupBy("Country").distinct("Brand").groupBy("Country").count()
val (xs, ys) = distinctBrandsPerCountry.rows.map{ row -> (row["Country"] as String) to (row["n"] as Int) }.unzip()
val p = lets_plot(mapoff("country" to xs, "brands" to ys))
|> val layer = geom_bar (stat=Stat.identity, fill = "#7B3CA") {
    x = "country"
    y = "brands"
}
p + layer
```

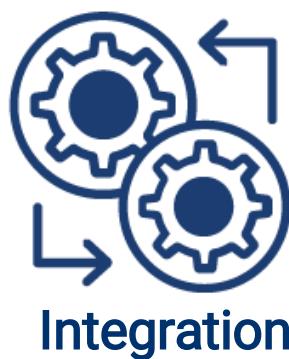
The output of cell [3] is a bar chart titled "Out[3]:" showing the number of brands per country. The x-axis is labeled "country" and the y-axis is labeled "brands". The chart shows that Japan has the highest number of brands (approximately 55), followed by the United States (approximately 45), and so on.

Country	Brands
Japan	~55
United States	~45
China	~30
United Kingdom	~25
South Korea	~20
Thailand	~18
Vietnam	~15
Malaysia	~12
Indonesia	~10
China	~10
Hong Kong	~8
Macau	~5
Philippines	~5
Bangladesh	~3
Canada	~3
Malta	~2
Netherlands	~2
U.S. Virgin Islands	~2
U.S. Commonwealths and Territories	~2
Dubai	~1

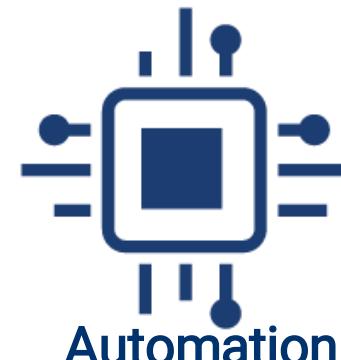
SYSTEMA GmbH

Art of Automation

“Industrie 4.0” made with ❤



Integration



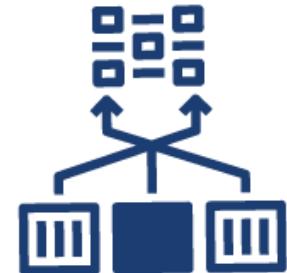
Automation



Optimization



Visualization



Migration



WE
ARE
HIRING

A graphic featuring five stylized human silhouettes against a dark blue background. From left to right: a man in a light blue hoodie; a woman in a dark blue blazer over a white shirt; a woman with orange hair in a dark blue dress; a woman in a dark blue dress with a belt; and a man in a light blue shirt and tie.

Join Us → career@systema.com

- Process Analysis & Modelling
- Scheduling & AI (Anomalies, RL, Regression)
- Factory Physics & Industrial Engineering
- Software Development (Java, Python, C#, SQL,...)
- BigData (Spark, Kafka, Elastic, Cloud) & BI
- Real-world problems in different manufacturing industries

- What we offer:
- Permanent contracts
 - Individual development plans
 - Individual benefits
 - Werkstudenten
 - Thesis projects



Thank you! Stay in touch!

[@holgerbrandl](#)

