raku::Dan
Gradual Types for Pandas DataFrames

our Raku 🎈 Journey:
scientist/PROGRAMMERS
DataFrame

* DataFrame is a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. (Pandas docs)
Top-level raku Data Analysis Module that provides a base set of raku-style datatype roles, accessors & methods, primarily:

- Series
- DataFrames
- A common basis for language / library bindings ...
  - Dan::Pandas (via Inline::Python),
  - Dan::Polars (via NativeCall / Rust FFI) [wip]

It's rather a zen concept since raku contains many Data Analysis constructs & concepts natively anyway.
raku Data Analytics Options

- Faster
- Library APIs & Tools
- Dan::Polars
- Dan::Pandas
- Typical Workflow
- More Flexible
- Pure raku
- Anton Power
- Data::Reshapers
- Data::Generators
- ML Oriented
More, Better Stuff...

- Data Munging | Wrangling
- Capture | Cleaning | Manipulation
- Data Contracts
- Validation | Gradual Types
Data Munging/Wrangling Aspects

* **Data discovery**
  This all-encompassing term describes how to understand your data. This is the first step to familiarize yourself with your data.

* **Structuring**
  The next step is to organize the data. Raw data is typically unorganized and much of it may not be useful for the end product. This step is important for easier computation and analysis in the later steps.

* **Cleaning**
  There are many different forms of cleaning data, for example one form of cleaning data is catching dates formatted in a different way and another form is removing outliers that will skew results and also formatting null values. This step is important in assuring the overall quality of the data.

* **Enriching**
  At this step determine whether or not additional data would benefit the data set that could be easily added.

* **Validating**
  This step is similar to structuring and cleaning. Use repetitive sequences of validation rules to assure data consistency as well as quality and security. An example of a validation rule is confirming the accuracy of fields via cross checking data.

* **Publishing**
  Prepare the data set for use downstream, which could include use for users or software. Be sure to document any steps and logic during wrangling.

raku Type system

https://docs.raku.org/images/type-graph-Numeric.svg
raku Type system

- Everything is a type
- Types are extensible
- Type composition
- Abstract types
- Subset / where
- Operator overloading
- Optional / gradual
- Type coercion
- Signatures / multis
- Mixins
- Regexes
- Allomorphs
Gradual Type Use Cases

- **Data discovery**
  Start with (Any), or maybe `read_csv()` is (Str). Avoid early commitment to specific types. In Raku you can add `[*]` Strs. Consider Allomorphs (IntStr or NumStr) that are closer to source data and can be specialized later. Use Raku to DWIM.

- **Structuring**
  Use types to "right-size" structure. Maybe a number series can be stored as Real (Int | Rat | Num). Check / coerce / round to e.g. all Int => int32 later when all sources are available and the data model is better defined. Use Rats for better decimals.

- **Cleaning**
  Combination of Raku 2nd generation regexes, look ahead type coercion and customized types great way to automatically & consistently cleanup.

- **Enriching**
  A scripting language with a diverse web ecosystem and CLI / database support, Raku helps to gather and match data feeds.

- **Validating**
  Validation rules and relationships can be implemented via (e.g.) custom bounds checking built into a structured set of custom types for easy maintenance.

- **Publishing**
  Raku Dan supports both Pandas and Polars (and their related plotting and environment tools) for speed, breadth and compatibility. By this point the dtypes need to be tightly defined (say int32) with proper NaN support.
OK - let's see it!

Using Brian Duggan's raku kernel for Jupyter on AWS Lightsail

https://github.com/p6steve/raku-Dan-Jupyter
container-service-1

Container service

Micro (1 GB RAM, 0.25 vCPUs) × 1 node
London

Status: Running

Public domain: container-service-1.gs7k0bfrb4gvk.eu-west-2.cs.amazonlightsail.com
How do I use my domain with my container service?
Private domain: container-service-1.service.local

Current deployment

Your deployment is the set of containers currently running on your container service.
Learn more about deployments

Modify your deployment

Deployed: May 17, 2022 - 9:36 AM

CONTAINERS
Public endpoint

raku-dan-pandas
Image: p6steve/raku-dan:jupyter-2022.02-amd64

PUBLIC ENDPOINT
raku-dan-pandas

<table>
<thead>
<tr>
<th>Port</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>8888</td>
<td>HTTP</td>
</tr>
</tbody>
</table>

Health check path
/

More details
### 1950s

<table>
<thead>
<tr>
<th>Year</th>
<th>Photos of winners</th>
<th>Role(s)</th>
<th>Film</th>
</tr>
</thead>
<tbody>
<tr>
<td>1954</td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/b/bf/Meena_Kumar.jpg/95px-Meena_Kumar.jpg" alt="Meena Kumar" /></td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/0/0d/Gauri.jpg/95px-Gauri.jpg" alt="Gauri" /></td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/1/10/Baiju_Bawra.png/95px-Baiju_Bawra.png" alt="Baiju Bawra (film)" /></td>
</tr>
<tr>
<td>1955</td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/b/bf/Meena_Kumar.jpg/95px-Meena_Kumar.jpg" alt="Meena Kumar" /></td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/4/4f/Sudha.jpg/95px-Sudha.jpg" alt="Sudha" /></td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/1/10/Baiju_Bawra.png/95px-Baiju_Bawra.png" alt="Baiju Bawra (film)" /></td>
</tr>
<tr>
<td>1956</td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/6/6c/Geeta_Dutt.jpg/95px-Geeta_Dutt.jpg" alt="Geeta Dutt" /></td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/2/2e/Kamala.jpg/95px-Kamala.jpg" alt="Kamala" /></td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/8/8e/Bengal_Tiger_%28film%29.png/95px-Bengal_Tiger_%28film%29.png" alt="Bengal Tiger (film)" /></td>
</tr>
<tr>
<td>1957</td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/b/bf/Meena_Kumar.jpg/95px-Meena_Kumar.jpg" alt="Meena Kumar" /></td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/0/0d/Gauri.jpg/95px-Gauri.jpg" alt="Gauri" /></td>
<td><img src="https://upload.wikimedia.org/wikipedia/commons/thumb/1/10/Baiju_Bawra.png/95px-Baiju_Bawra.png" alt="Baiju Bawra (film)" /></td>
</tr>
</tbody>
</table>
PART I: USE REGEXES TO EXTRACT DATA ITEMS

my @exclusions = <Year|Photos|Actress|Role|Film|Filmfare|Awards|File|No|Other|Nominee|Portrait|Returns>;

my regex tcword { # title case word
    <upper>       # first character uppercase
    <lower>++     # rest of chars lowercase
}

my regex name { # one or more separated by a space
    <tcword>+ % % ' ' 
}

my regex year { # four consecutive digits
    \d{4}
}

my ($current, @years, @names);

for @lines -> $l is rw {
    next if $l ~~ /File/; # avoid getting name from photo links
    $l ~~ s:g/<@exclusions>//; # strip out exclusions
    if $l ~~ /<year>/ {
        $current = $year; # current year is 'sticky'
    }
    if $l ~~ /<name>/ { #
        @years.push: $current;
        @names.push: ~<$name>.trim;
    }
} @names[0..10].join(":\: ").say;

Meena::Gauri::Baiju Bawra::Meena::Lalita::Parineeta::Kamini::Biraj::Biraj Bahu::Geeta::Kamala

2nd generation raku
Regexes provide maintainable matching power

From the team that brought you PCRE
### PART II: MAP DATA ITEMS TO 2-D ARRAY AND MAKE DATAFRAME

```perl
my @name3 = @names.rotor(3);  # rotor name over cols 1..3
my @year3 = @years.rotor(3);  # match year cadence to names
my @data;                     # 2d array

for 0..@year3 -> $m {
    next unless @year3[0][0];
    @data[$m;0] = @year3[0][0];
    @data[$m;1..3] = @name3[$m];
}

my @columns = <Year Actor Role Film>;

my \df = DataFrame.new( :@data, :@columns );
say ~df[0..5];
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Actor</th>
<th>Role</th>
<th>Film</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1954</td>
<td>Meena</td>
<td>Gauri</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Baiju Bawra</td>
</tr>
<tr>
<td>1</td>
<td>1955</td>
<td>Meena</td>
<td>Lalita</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Parineeta</td>
</tr>
<tr>
<td>2</td>
<td>1956</td>
<td>Kamini</td>
<td>Biraj</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Biraj Bahu</td>
</tr>
<tr>
<td>3</td>
<td>1956</td>
<td>Geeta</td>
<td>Kamala</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Vachan</td>
</tr>
<tr>
<td>4</td>
<td>1956</td>
<td>Meena</td>
<td>Shobha</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Azaad</td>
</tr>
<tr>
<td>5</td>
<td>1957</td>
<td>Nutan</td>
<td>Gauri</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Seema</td>
</tr>
</tbody>
</table>
PART III: MAKE SOME NEW SERIES AND DATAFRAMES TYPES

class YearSeries {
    has Series $.series handles *;

    method TWEAK {
        given $.data {
            unless ( 1
                && .all ~ /<year>/
                && 1950 < .all < 2030
                &&& 1940 < .all < 2030
            ) {
                die "Data fails to meet {$.self.^name} constraint."
            }
        }
    }
}

class NameSeries {
    has Series $.series handles *;

    method TWEAK {
        given $.data {
            unless (.all ~ /<name>/ ) {
                die "Data fails to meet {$.self.^name} constraint."
            }
        }
    }
}
class FilmDataFrame {
    has DataFrame $dataframe handles *;

    method TWEAK {
        unless ( 1
            && YearSeries.new(series => self.dataframe[*]<Year>)
            && NameSeries.new(series => self.dataframe[*]<Actor>)
            && NameSeries.new(series => self.dataframe[*]<Role>)
            && NameSeries.new(series => self.dataframe[*]<Film>)
        ) {
            die "Data fails to meet {$.self.^name} constraint."
        }
    }
}

say 'ok'

ok

PART IV: PERFORM TYPE CHECKS AT SERIES AND DATAFRAME LEVEL

my FilmDataFrame $fdf = new( dataframe => df );

Data fails to meet YearSeries constraint.

say df[*]<Year>.grep(* <= 1950);

([1942])
raku: a new frontier

we (yes we!) can start with a clean sheet
anyone / everyone is welcome to join in
PART II: MAP DATA ITEMS TO 2-D ARRAY AND MAKE DATAFRAME

```perl
my @name3 = @names.rotor(3);  # rotor name over cols 1..3
my @year3 = @years.rotor(3);  # match year cadence to names
my @data;                      # 2d array

for 0..@year3 -> $m {
    next unless @year3[$m][0];
    @data[$m;0] = @year3[$m][0];
    @data[$m;1..3] = @name3[$m];
}

my @columns = <Year Actor Role Film>;

my \df = DataFrame.new( :@data, :@columns );
say ^df[0..5]^;
```

<table>
<thead>
<tr>
<th>Year</th>
<th>Actor</th>
<th>Role</th>
<th>Film</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Meena</td>
<td>Gauri</td>
<td>Baiju Bawra</td>
</tr>
<tr>
<td>1</td>
<td>Meena</td>
<td>Lalita</td>
<td>Parineeta</td>
</tr>
<tr>
<td>2</td>
<td>Kamini</td>
<td>Biraj</td>
<td>Biraj Bahu</td>
</tr>
<tr>
<td>3</td>
<td>Geeta</td>
<td>Kamala</td>
<td>Vachan</td>
</tr>
<tr>
<td>4</td>
<td>Meena</td>
<td>Shobha</td>
<td>Azaad</td>
</tr>
<tr>
<td>5</td>
<td>Nutan</td>
<td>Gauri</td>
<td>Seema</td>
</tr>
</tbody>
</table>
PART III: HISTOGRAM FOR OUTLIERS

```r
my \abh = df[\*]<Actor>.Bag;
abh.^name;

Bag

for abh.sort(-*.value).[0..^40] { say .key.fmt('%-20s'), '+' xx .value }
```

Meena
(+ + + + + + + + +)
Rani
(+ + + + + + + +)
Aishwarya
(+ + + + + + + + +)
Aarti
(+ + + + +)
Preity
(+ + + + +)
Karisma
(+ + + +)
Kareena
(+ + + +)
Vidya
(+ + + +)
Kajol
(+ + + +)
Urmila
(+ + +)
Vyjayanthimala
(+ + +)
Tabu
(+ + +)
Hema
(+ + +)
Nutan
(+ + +)
Madhu
(+ + +)
Madhuri
(+ + +)
PART IV: TEXT PLOT FOR OUTLIERS

zef install Text::Plot from Anton Antonov (nice!)

my \ybh = df[*]<Year>.Bag;
ybh.^name;

Bag

use Text::Plot;

say text-list-plot([ybh.keys.map(++)], [ybh.values],
    point-char => '+',
    x-tick-labels-format => '%d',
    y-tick-labels-format => '%d',
);

* Text::Plot for quick &
dirty visualization
PART V: SET MATH

my \texttt{aset} \texttt{=} \texttt{df[*]<Actor>.Set;}
my \texttt{rset} \texttt{=} \texttt{df[*]<Role>.Set;}
my \texttt{fset} \texttt{=} \texttt{df[*]<Film>.Set;}
say 'ok'

ok

'Bipasha' \texttt{\in} \texttt{aset} \hspace{1cm} \# set element

True

my \texttt{rset} \texttt{=} \texttt{aset} \texttt{\&} \texttt{rset} \texttt{\&} \texttt{fset} \hspace{1cm} \# set intersect (names in Actors & Roles & Films)


Make an inverse look up for the years each of these won

# first we make an inverse look up
my \texttt{bactor} \texttt{=} \texttt{%\{given \texttt{df[*]<Actor> \{ .data Z => .index.values \}}\}};

# then we can iterate the set keys through it
\{ \texttt{df[bactor[\$k]<Year>} \texttt{for xset.keys} \}

([2017] [1973] [2017] [1980] [2002] [1983] [2010] [1992] [2007] [2014] [2001] [1978] [1980] [1973] [1992] [1972] [1997] [2010] [1979] [2005])
# Distribution:

```python
vis1 = sns.distplot(stats['InternetUsers'], bins=100)
```

# Boxplots:

```python
vis2 = sns.boxplot(data=stats, x="IncomeGroup", y="BirthRate")
```
role DataSlice {
    has Str $.name;
    has Any @.data;
    has Int %.index;
}

role Series does DataSlice {
    method dtype { @.data.are }
}

role DataFrame {
    has Any @.data =[];  #2d [m;n]
    has Int %.index;    #row index
    has Int %.columns;  #col index

    method series( $k ) { $.[*]{$k} }

    method dtypes {
        $.columns.&sbv.map({$.series($_)})
            .map( *.dtype )
    }
}
What are we getting from raku core that others do in libraries?

- pipes & maps
- multi-dimensional arrays
- slicing & indexing
- references & views
- map, reduce, hyper operators
- operator overloading
- concurrency, supplies
- types (incl. NaN)
Dan roadmap

Dan Base (+AST?)

Dan::Mezza

User Code

"::Array+"

::Query ::Sheet ::Set ::Math ::Regex

Dan::Polars (NativeCall)

Dan::Pandas (Inline::Python)

Dan::PDL (perl PDL?)

* * *

* * *

=> taking shape

=> prototype

=> open season

p6steve.com ♊
method count { $.elems }

method mean { $.sum / $.elems }

method std {
    sqrt ( [+] $.data.map({ $^x - $.mean })
             .map({ $^x ** 2 }) / ( $.elems - 1 ) )
}
Please Help!

* Range of skill sets (you can help)
  * Starter - write Mezza methods (see previous) & blog examples
  * Intermediate - improved Dan::Pandas workflow / matplotlib, seaborn modules, Git Actions, Acquisition Regexes...
  * Experienced - Dan AST, Polars native types, shaped Arrays

* Support & guidance provided
* Pandas "experimenters" welcome!!

mailto: p6steve@furnival.net
Thanks! Questions?
Useful Links

* [https://github.com/p6steve/raku-Dan](https://github.com/p6steve/raku-Dan)
* [https://github.com/p6steve/raku-Dan-Pandas](https://github.com/p6steve/raku-Dan-Pandas)
* [https://github.com/bduggan/p6-jupyter-kernel](https://github.com/bduggan/p6-jupyter-kernel)
* [https://lightsail.aws.amazon.com](https://lightsail.aws.amazon.com)
* [https://github.com/p6steve/raku-Dan-Jupyter](https://github.com/p6steve/raku-Dan-Jupyter)