Proposed API for tech.ml.dataset

GenerateMe

2020-05-20

Introduction

tech.ml.dataset is a great and fast library which brings columnar dataset to the Clojure. Chris Nuernberger has been working on this library for last year as a part of bigger `tech.ml` stack.

I've started to test the library and help to fix uncovered bugs. My main goal was to compare functionalities with the other standards from other platforms. I focused on R solutions: dplyr, tidyr and data.table.

During conversions of the examples I've come up how to reorganized existing `tech.ml.dataset` functions into simple to use API. The main goals were:

- Focus on dataset manipulation functionality, leaving other parts of `tech.ml` like pipelines, datatypes, readers, ML, etc.
- Single entry point for common operations - one function dispatching on given arguments.
- `group-by` results with special kind of dataset - a dataset containing subsets created after grouping as a column.
- Most operations recognize regular dataset and grouped dataset and process data accordingly.
- One function form to enable thread-first on dataset.

All proposed functions are grouped in tabs below. Select group to see examples and details.

INFO: The future of this API is not known yet. Two directions are possible: integration into `tech.ml` or development under `Scicloj` organization. For the time being use this repo if you want to try. Join the discussion on Zulip

Let's require main namespace and define dataset used in most examples:

```
(require '[techtest.api :as api])
(def DS (api/dataset {:V1 (take 9 (cycle [1 2]))
                      :V2 (range 1 10)
                      :V3 (take 9 (cycle [0.5 1.0 1.5]))
                      :V4 (take 9 (cycle [\A \B \C]))}))
```

```
DS
```

_unnamed [9 4]:

:V1	:V2	:V3	:V4
1	1	0.5000	A
2	2	1.000	B
1	3	1.500	C
2	4	0.5000	A
1	5	1.000	B
2	6	1.500	C
1	7	0.5000	A
2	8	1.000	B

:V1	:V2	:V3	:V4
1	9	1.500	C

Functionality

Dataset

Dataset is a special type which can be considered as a map of columns implemented around `tech.ml.datatype` library. Each column can be considered as named sequence of typed data. Supported types include integers, floats, string, boolean, date/time, objects etc.

Dataset creation

Dataset can be created from various of types of Clojure structures and files:

- single values
- sequence of maps
- map of sequences or values
- sequence of columns (taken from other dataset or created manually)
- sequence of pairs
- file types: raw/gzipped csv/tsv, json, xls(x) taken from local file system or URL
- input stream

`api/dataset` accepts:

- data
- options (see documentation of `tech.ml.dataset/->dataset` function for full list):
- `:dataset-name` - name of the dataset
- `:num-rows` - number of rows to read from file
- `:header-row?` - indication if first row in file is a header
- `:key-fn` - function applied to column names (eg. `keyword`, to convert column names to keywords)
- `:separator` - column separator
- `:single-value-column-name` - name of the column when single value is provided

Empty dataset.

```
(api/dataset)
```

```
_unnamed [0 0]:
```

Dataset from single value.

```
(api/dataset 999)
```

_unnamed [1 1]:

:$value
999

Set column name for single value. Also set the dataset name.

```
(api/dataset 999 {:single-value-column-name "my-single-value"})
(api/dataset 999 {:single-value-column-name ""
                  :dataset-name "Single value"})
```

_unnamed [1 1]:

my-single-value
999

Single value [1 1]:

0
999

Sequence of pairs (first = column name, second = value(s)).

```
(api/dataset [[:A 33] [:B 5] [:C :a]])
```

_unnamed [1 3]:

:A	:B	:C
33	5	:a

Not sequential values are repeated row-count number of times.

```
(api/dataset [[:A [1 2 3 4 5 6]] [:B "X"] [:C :a]])
```

_unnamed [6 3]:

:A	:B	:C
1	X	:a
2	X	:a
3	X	:a
4	X	:a
5	X	:a
6	X	:a

Dataset created from map (keys = column name, second = value(s)). Works the same as sequence of pairs.

```
(api/dataset {:A 33})
(api/dataset {:A [1 2 3]})
(api/dataset {:A [3 4 5] :B "X"})
```

_unnamed [1 1]:

:A
33

_unnamed [3 1]:

:A
1
2
3

_unnamed [3 2]:

:A	:B
3	X
4	X
5	X

You can put any value inside a column

```
(api/dataset {:A [[3 4 5] [:a :b]] :B "X"})
```

_unnamed [2 2]:

:A	:B
[3 4 5]	X
[:a :b]	X

Sequence of maps

```
(api/dataset [{:a 1 :b 3} {:b 2 :a 99}])
(api/dataset [{:a 1 :b [1 2 3]} {:a 2 :b [3 4]}])
```

_unnamed [2 2]:

:a	:b
1	3
99	2

_unnamed [2 2]:

:a	:b
1	[1 2 3]
2	[3 4]

Missing values are marked by `nil`

```
(api/dataset [{:a nil :b 1} {:a 3 :b 4} {:a 11}])
```

_unnamed [3 2]:

:a	:b
	1
3	4
11	

Import CSV file

```
(api/dataset "data/family.csv")
```

data/family.csv [5 5]:

family	dob_child1	dob_child2	gender_child1	gender_child2
1	1998-11-26	2000-01-29	1	2
2	1996-06-22		2	
3	2002-07-11	2004-04-05	2	2
4	2004-10-10	2009-08-27	1	1
5	2000-12-05	2005-02-28	2	1

Import from URL

```
(defonce ds (api/dataset "https://vega.github.io/vega-lite/examples/data/seattle-weather.csv"))
```

```
ds
```

https://vega.github.io/vega-lite/examples/data/seattle-weather.csv [1461 6]:

date	precipitation	temp_max	temp_min	wind	weather
2012-01-01	0.000	12.80	5.000	4.700	drizzle
2012-01-02	10.90	10.60	2.800	4.500	rain
2012-01-03	0.8000	11.70	7.200	2.300	rain
2012-01-04	20.30	12.20	5.600	4.700	rain
2012-01-05	1.300	8.900	2.800	6.100	rain
2012-01-06	2.500	4.400	2.200	2.200	rain
2012-01-07	0.000	7.200	2.800	2.300	rain
2012-01-08	0.000	10.00	2.800	2.000	sun
2012-01-09	4.300	9.400	5.000	3.400	rain
2012-01-10	1.000	6.100	0.6000	3.400	rain
2012-01-11	0.000	6.100	-1.100	5.100	sun
2012-01-12	0.000	6.100	-1.700	1.900	sun
2012-01-13	0.000	5.000	-2.800	1.300	sun
2012-01-14	4.100	4.400	0.6000	5.300	snow
2012-01-15	5.300	1.100	-3.300	3.200	snow
2012-01-16	2.500	1.700	-2.800	5.000	snow
2012-01-17	8.100	3.300	0.000	5.600	snow

date	precipitation	temp_max	temp_min	wind	weather
2012-01-18	19.80	0.000	-2.800	5.000	snow
2012-01-19	15.20	-1.100	-2.800	1.600	snow
2012-01-20	13.50	7.200	-1.100	2.300	snow
2012-01-21	3.000	8.300	3.300	8.200	rain
2012-01-22	6.100	6.700	2.200	4.800	rain
2012-01-23	0.000	8.300	1.100	3.600	rain
2012-01-24	8.600	10.00	2.200	5.100	rain
2012-01-25	8.100	8.900	4.400	5.400	rain

Saving

Export dataset to a file or output stream can be done by calling `api/write-csv!`. Function accepts:

- dataset
- file name with one of the extensions: `.csv`, `.tsv`, `.csv.gz` and `.tsv.gz` or output stream
- options:
- `:separator` - string or separator char.

```
(api/write-csv! ds "output.tsv.gz")
(.exists (clojure.java.io/file "output.csv.gz"))
```

```
nil
true
```

Dataset related functions

Summary functions about the dataset like number of rows, columns and basic stats.

Number of rows

```
(api/row-count ds)
```

```
1461
```

Number of columns

```
(api/column-count ds)
```

```
6
```

Names of columns.

```
(api/column-names ds)
```

```
("date" "precipitation" "temp_max" "temp_min" "wind" "weather")
```

Shape of the dataset, [row count, column count]

```
(api/shape ds)
```

```
[1461 6]
```

General info about dataset. There are three variants:

- default - containing information about columns with basic statistics
- `:basic` - just name, row and column count and information if dataset is a result of `group-by` operation
- `:columns` - columns' metadata

```
(api/info ds)
(api/info ds :basic)
(api/info ds :columns)
```

https://vega.github.io/vega-lite/examples/data/seattle-weather.csv: descriptive-stats [6 10]:

:col-name	:datatype	:n-valid	:n-missing	:mean	:mode	:min	:max	:standard-deviation	:skew
date	:packed-local-date	1461	0	2013-12-31		2012-01-01	2015-12-31		
precipitation	:float32	1461	0	3.029		0.000	55.90	6.680	3.506
temp_max	:float32	1461	0	16.44		-1.600	35.60	7.350	0.2809
temp_min	:float32	1461	0	8.235		-7.100	18.30	5.023	-0.2495
weather	:string	1461	0		sun				
wind	:float32	1461	0	3.241		0.4000	9.500	1.438	0.8917

https://vega.github.io/vega-lite/examples/data/seattle-weather.csv :basic info [1 4]:

:name	:grouped?	:rows	:columns
https://vega.github.io/vega-lite/examples/data/seattle-weather.csv	false	1461	6

https://vega.github.io/vega-lite/examples/data/seattle-weather.csv :column info [6 4]:

:name	:size	:datatype	:categorical?
date	1461	:packed-local-date	
precipitation	1461	:float32	
temp_max	1461	:float32	
temp_min	1461	:float32	
wind	1461	:float32	
weather	1461	:string	true

Getting a dataset name

```
(api/dataset-name ds)
```

```
"https://vega.github.io/vega-lite/examples/data/seattle-weather.csv"
```

Setting a dataset name (operation is immutable).

```
(->> "seattle-weather"
     (api/set-dataset-name ds)
     (api/dataset-name))
```

```
"seattle-weather"
```

Columns and rows

Get columns and rows as sequences. `column`, `columns` and `rows` treat grouped dataset as regular one. See `Groups` to read more about grouped datasets.

Select column.

```
(ds "wind")
(api/column ds "date")
```

```
#tech.ml.dataset.column<float32>[1461]
wind
[4.700, 4.500, 2.300, 4.700, 6.100, 2.200, 2.300, 2.000, 3.400, 3.400, 5.100, 1.900, 1.300, 5.300, 3.20
#tech.ml.dataset.column<packed-local-date>[1461]
date
[2012-01-01, 2012-01-02, 2012-01-03, 2012-01-04, 2012-01-05, 2012-01-06, 2012-01-07, 2012-01-08, 2012-0
```

Columns as sequence

```
(take 2 (api/columns ds))
```

```
(#tech.ml.dataset.column<packed-local-date>[1461]
date
[2012-01-01, 2012-01-02, 2012-01-03, 2012-01-04, 2012-01-05, 2012-01-06, 2012-01-07, 2012-01-08, 2012-0
precipitation
[0.000, 10.90, 0.8000, 20.30, 1.300, 2.500, 0.000, 0.000, 4.300, 1.000, 0.000, 0.000, 0.000, 4.100, 5.3
```

Columns as map

```
(keys (api/columns ds :as-map))
```

```
("date" "precipitation" "temp_max" "temp_min" "wind" "weather")
```

Rows as sequence of sequences

```
(take 2 (api/rows ds))
```

```
([#object[java.time.LocalDate 0x5870656 "2012-01-01"] 0.0 12.8 5.0 4.7 "drizzle"] [#object[java.time.Lo
```

Rows as sequence of maps

```
(clojure.pprint/pprint (take 2 (api/rows ds :as-maps)))
```

```
({"date" #object[java.time.LocalDate 0x109a18b7 "2012-01-01"],
  "precipitation" 0.0,
  "temp_min" 5.0,
```

```
  "weather" "drizzle",
  "temp_max" 12.8,
  "wind" 4.7}
 {"date" #object[java.time.LocalDate 0x49d2bfb5 "2012-01-02"],
  "precipitation" 10.9,
  "temp_min" 2.8,
  "weather" "rain",
  "temp_max" 10.6,
  "wind" 4.5})
```

Group-by

Grouping by is an operation which splits dataset into subdatasets and pack it into new special type of... dataset. I distinguish two types of dataset: regular dataset and grouped dataset. The latter is the result of grouping.

Grouped dataset is annotated in by `:grouped?` meta tag and consist following columns:

- `:name` - group name or structure
- `:group-id` - integer assigned to the group
- `:count` - number of elements in a group
- `:data` - groups as datasets

Almost all functions recognize type of the dataset (grouped or not) and operate accordingly.

You can't apply reshaping or join/concat functions on grouped datasets.

Grouping

Grouping is done by calling `group-by` function with arguments:

- `ds` - dataset
- `grouping-selector` - what to use for grouping
- options:
- `:result-type` - what to return:
- `:as-dataset` (default) - return grouped dataset
- `:as-indexes` - return rows ids (row number from original dataset)
- `:as-map` - return map with group names as keys and subdataset as values
- `:limit-columns` - list of the columns which should be returned during grouping by function.

Grouping can be done by:

- single column name
- seq of column names
- map of keys (group names) and row indexes
- value returned by function taking row as map

Note: currently dataset inside dataset is printed recursively so it renders poorly from markdown.

List of columns in groupd dataset

```
(api/column-names (api/group-by DS :V1))
```

```
(:name :group-id :count :data)
```

Content of the grouped dataset

```
(api/columns (api/group-by DS :V1) :as-map)
```

```
{:name #tech.ml.dataset.column<int64>[2]
:name
[1, 2, ], :group-id #tech.ml.dataset.column<int64>[2]
:group-id
[0, 1, ], :count #tech.ml.dataset.column<int32>[2]
:count
[5, 4, ], :data #tech.ml.dataset.column<object>[2]
:data
[1 [5 4]:

| :V1 | :V2 |    :V3 | :V4 |
|-----+-----+--------+-----|
|   1 |   1 | 0.5000 |   A |
|   1 |   3 |  1.500 |   C |
|   1 |   5 |  1.000 |   B |
|   1 |   7 | 0.5000 |   A |
|   1 |   9 |  1.500 |   C |
, 2 [4 4]:

| :V1 | :V2 |    :V3 | :V4 |
|-----+-----+--------+-----|
|   2 |   2 |  1.000 |   B |
|   2 |   4 | 0.5000 |   A |
|   2 |   6 |  1.500 |   C |
|   2 |   8 |  1.000 |   B |
, ]}
```

Grouped dataset as map

```
(keys (api/group-by DS :V1 {:result-type :as-map}))
```

```
(1 2)
```

```
(vals (api/group-by DS :V1 {:result-type :as-map}))
```

(_unnamed [5 4]:

:V1	:V2	:V3	:V4
1	1	0.5000	A
1	3	1.500	C
1	5	1.000	B
1	7	0.5000	A
1	9	1.500	C

_unnamed [4 4]:

:V1	:V2	:V3	:V4
2	2	1.000	B
2	4	0.5000	A

:V1	:V2	:V3	:V4
2	6	1.500	C
2	8	1.000	B

)

Group dataset as map of indexes (row ids)

```
(api/group-by DS :V1 {:result-type :as-indexes})
```

```
{1 [0 2 4 6 8], 2 [1 3 5 7]}
```

Ungrouping

Other functions

Columns

Rows

Aggregate

Order

Unique

Missing

Join/Split Columns

Fold/Unroll Rows

Reshape

Join/Concat