Frames: Data frames For working with tabular data files

[bsd3, data, library] [Propose Tags]

User-friendly, type safe, runtime efficient tooling for working with tabular data deserialized from comma-separated values (CSV) files. The type of each row of data is inferred from data, which can then be streamed from disk, or worked with in memory.

[Skip to Readme]

Build Instaforok Documentation Available

Modules

[Index] [Quick Jump]

Frames
- Frames.CSV
- Frames.Categorical
- Frames.Col
- Frames.ColumnTypeable
- Frames.ColumnUniverse
- Frames.Exploration
- Frames.ExtraInstances
- Frames.Frame
- Frames.InCore
- Frames.Joins
- Frames.Melt
- Frames.Rec
- Frames.RecF
- Frames.ShowCSV
- Frames.TH
- Frames.TypeLevel
- Frames.Utils

Manual Flags

Name	Description	Default
demos	Build demonstration programs	Disabled

▶ Automatic Flags

Use `-f <flag>` to enable a flag, or `-f -<flag>` to disable that flag. More info

Downloads

- Frames-0.7.2.tar.gz [browse] (Cabal source package)
- Package description (as included in the package)

Maintainer's Corner

For package maintainers and hackage trustees
- edit package information

Candidates
- No Candidates

Versions [RSS] [flat]
0.1.0.0, 0.1.1.0, 0.1.1.1, 0.1.2, 0.1.2.1, 0.1.3, 0.1.4, 0.1.6, 0.1.8, 0.1.9, 0.2.0, 0.2.1, 0.2.1.1, 0.3.0, 0.3.0.1, 0.3.0.2, 0.4.0, 0.5.0, 0.5.1, 0.6.0, 0.6.1, 0.6.2, 0.6.3, 0.6.4, 0.7.0, 0.7.1, **0.7.2**

Change log
CHANGELOG.md

Dependencies
base (>=4.10 && <4.16), bytestring, containers, contravariant, deepseq (>=1.4), discrimination, ghc-prim (>=0.3 && <0.8), hashable, pipes (>=4.1 && <5), pipes-bytestring (>=2.1.6 && <2.2), pipes-group (>=1.0.8 && <1.1), pipes-parse (>=3.0.*), pipes-safe (>=2.2.6 && <2.4), primitive (>=0.6 && <0.8), readable (>=0.3.1), template-haskell, text (>=1.1.1.0), transformers, vector, vector-th-unbox (>=0.2.1.6), vinyl (>=0.13.0 && <0.14) [details]

License
BSD-3-Clause

Copyright
Copyright (C) 2014-2018 Anthony Cowley

Author
Anthony Cowley

Maintainer
acowley@gmail.com

Category
Data

Source repo
head: git clone http://github.com/acowley/Frames.git

Uploaded
by AnthonyCowley at 2021-05-19T16:47:00Z

Distributions
NixOS:0.7.2, Stackage:0.7.2

Executables
modcsv, kata04, missing, benchdemo, tutorial, demo, plot2, plot, getdata

Downloads
17124 total (47 in the last 30 days)

Rating
2.0 (votes: 1) [estimated by Bayesian average]

Your Rating
★ ★ ★

Status
Docs available [build log]
Last success reported on 2021-05-19 [all 1 reports]

Readme for Frames-0.7.2

[back to package description]

Frames

Data Frames for Haskell

User-friendly, type safe, runtime efficient tooling for working with tabular data deserialized from comma-separated values (CSV) files. The type of each row of data is inferred from data, which can then be streamed from disk, or worked with in memory.

We provide streaming and in-memory interfaces for efficiently working with datasets that can be safely indexed by column names found in the data files themselves. This type safety of column access and manipulation is checked at compile time.

Use Cases

For a running example, we will use variations of the prestige.csv data set. Each row includes 7 columns, but we just want to compute the average ratio of `income` to `prestige`.

Clean Data

If you have a CSV data where the values of each column may be classified by a single type, and ideally you have a header row giving each column a name, you may simply want to avoid writing out the Haskell type corresponding to each row. `Frames` provides `TemplateHaskell` machinery to infer a Haskell type for each row of your data set, thus preventing the situation where your code quietly diverges from your data.

We generate a collection of definitions generated by inspecting the data file at compile time (using `tableTypes`), then, at runtime, load that data into column-oriented storage in memory (an **in-core** array of structures (AoS)). We're going to compute the average ratio of two columns, so we'll use the `foldl` library. Our fold will project the columns we want, and apply a function that divides one by the other after appropriate numeric type conversions. Here is the entirety of that program.

```
{-# LANGUAGE DataKinds, FlexibleContexts, QuasiQuotes, TemplateHaskell, TypeApplications #-}
module UncurryFold where
import qualified Control.Foldl                as L
import           Data.Vinyl.Curry              ( runcurryX )
import           Frames

-- Data set from http://vincentarelbundock.github.io/Rdatasets/datasets.html
tableTypes "Row" "test/data/prestige.csv"

loadRows :: IO (Frame Row)
loadRows = inCoreAoS (readTable "test/data/prestige.csv")

-- | Compute the ratio of income to prestige for a record containing
-- only those fields.
ratio :: Record '[Income, Prestige] -> Double
ratio = runcurryX (\i p -> fromIntegral i / p)

averageRatio :: IO Double
averageRatio = L.fold (L.premap (ratio . rcast) avg) <$> loadRows
  where avg = (/) <$> L.sum <*> L.genericLength
```

Missing Header Row

Now consider a case where our data file lacks a header row (I deleted the first row from `prestige.csv`). We will provide our own name for the generated row type, our own column names, and, for the sake of demonstration, we will also specify a prefix to be added to every column-based identifier (particularly useful if the column names **do** come from a header row, and you want to work with multiple CSV files some of whose column names coincide). We customize behavior by updating whichever fields of the record produced by `rowGen` we care to change, passing the result to `tableTypes'`. Link to code.

```
{-# LANGUAGE DataKinds, FlexibleContexts, QuasiQuotes, TemplateHaskell, TypeApplications #-}
module UncurryFoldNoHeader where
import qualified Control.Foldl                as L
import           Data.Vinyl.Curry              ( runcurryX )
import           Frames
import           Frames.TH                     ( rowGen
                                               , RowGen(..)
                                               )

-- Data set from http://vincentarelbundock.github.io/Rdatasets/datasets.html
tableTypes' (rowGen "test/data/prestigeNoHeader.csv")
            { rowTypeName = "NoH"
            , columnNames = [ "Job", "Schooling", "Money", "Females"
                            , "Respect", "Census", "Category" ]
            , tablePrefix = "NoHead"}

loadRows :: IO (Frame NoH)
loadRows = inCoreAoS (readTableOpt noHParser "test/data/prestigeNoHeader.csv")

-- | Compute the ratio of money to respect for a record containing
-- only those fields.
ratio :: Record '[NoHeadMoney, NoHeadRespect] -> Double
ratio = runcurryX (\m r -> fromIntegral m / r)

averageRatio :: IO Double
averageRatio = L.fold (L.premap (ratio . rcast) avg) <$> loadRows
  where avg = (/) <$> L.sum <*> L.genericLength
```

Missing Data

Sometimes not every row has a value for every column. I went ahead and blanked the `prestige` column of every row whose `type` column was NA in `prestige.csv`. For example, the first such row now reads,

```
"athletes",11.44,8206,8.13,,3373,NA
```

We can no longer parse a `Double` for that row, so we will work with row types parameterized by a `Maybe` type constructor. We are substantially filtering our data, so we will perform this operation in a streaming fashion without ever loading the entire table into memory. Our process will be to check if the `prestige` column was parsed, only keeping those rows for which it was not, then project the `income` column from those rows, and finally throw away `Nothing` elements. Link to code.

```
{-# LANGUAGE DataKinds, FlexibleContexts, QuasiQuotes, TemplateHaskell, TypeApplications, TypeOp
module UncurryFoldPartialData where
import qualified Control.Foldl as L
import Data.Maybe (isNothing)
import Data.Vinyl.XRec (toHKD)
import Frames
import Pipes (Producer, (>->))
import qualified Pipes.Prelude as P

-- Data set from http://vincentarelbundock.github.io/Rdatasets/datasets.html
-- The prestige column has been left blank for rows whose "type" is
-- listed as "NA".
tableTypes "Row" "test/data/prestigePartial.csv"

-- | A pipes 'Producer' of our 'Row' type with a column functor of
-- 'Maybe'. That is, each element of each row may have failed to parse
-- from the CSV file.
maybeRows :: MonadSafe m => Producer (Rec (Maybe :. ElField) (RecordColumns Row)) m ()
maybeRows = readTableMaybe "test/data/prestigePartial.csv"

-- | Return the number of rows with unknown prestige, and the average
-- income of those rows.
incomeOfUnknownPrestige :: IO (Int, Double)
incomeOfUnknownPrestige =
  runSafeEffect . L.purely P.fold avg $
    maybeRows >-> P.filter prestigeUnknown >-> P.map getIncome >-> P.concat
  where avg = (\n l -> (l, s / fromIntegral l)) <$> L.sum <*> L.length
        getIncome = fmap fromIntegral . toHKD . rget @Income
        prestigeUnknown :: Rec (Maybe :. ElField) (RecordColumns Row) -> Bool
        prestigeUnknown = isNothing . toHKD . rget @Prestige
```

Tutorial

For comparison to working with data frames in other languages, see the tutorial.

Demos

There are various demos in the repository. Be sure to run the `getdata` build target to download the data files used by the demos! You can also download the data files manually and put them in a `data` directory in the directory from which you will be running the executables.

Benchmarks

The benchmark shows several ways of dealing with data when you want to perform multiple traversals.

Another demo shows how to fuse multiple passes into one so that the full data set is never resident in memory. A Pandas version of a similar program is also provided for comparison.

This is a trivial program, but shows that performance is comparable to Pandas, and the memory savings of a compiled program are substantial.

First with Pandas,

```
$ nix-shell -p 'python3.withPackages (p: [p.pandas])' --run '$(which time) -f "%Uuser %Ssystem %
28.087476512228815
-81.90356506136422
0.67user 0.04system 0:00.72elapsed 99%CPU; 79376maxresident KB
```

Then with Frames,

```
$ $(which time) -f '%Uuser %Ssystem %Eelapsed %PCPU; %Mmaxresident KB' dist-newstyle/build/x86_6
28.087476512228815
-81.90356506136422
0.36user 0.00system 0:00.37elapsed 100%CPU; 5088maxresident KB
```