DataFrames: The Extended Cut

Wes McKinney @wesmckinn

Open Data Science Conference, 2015-05-31

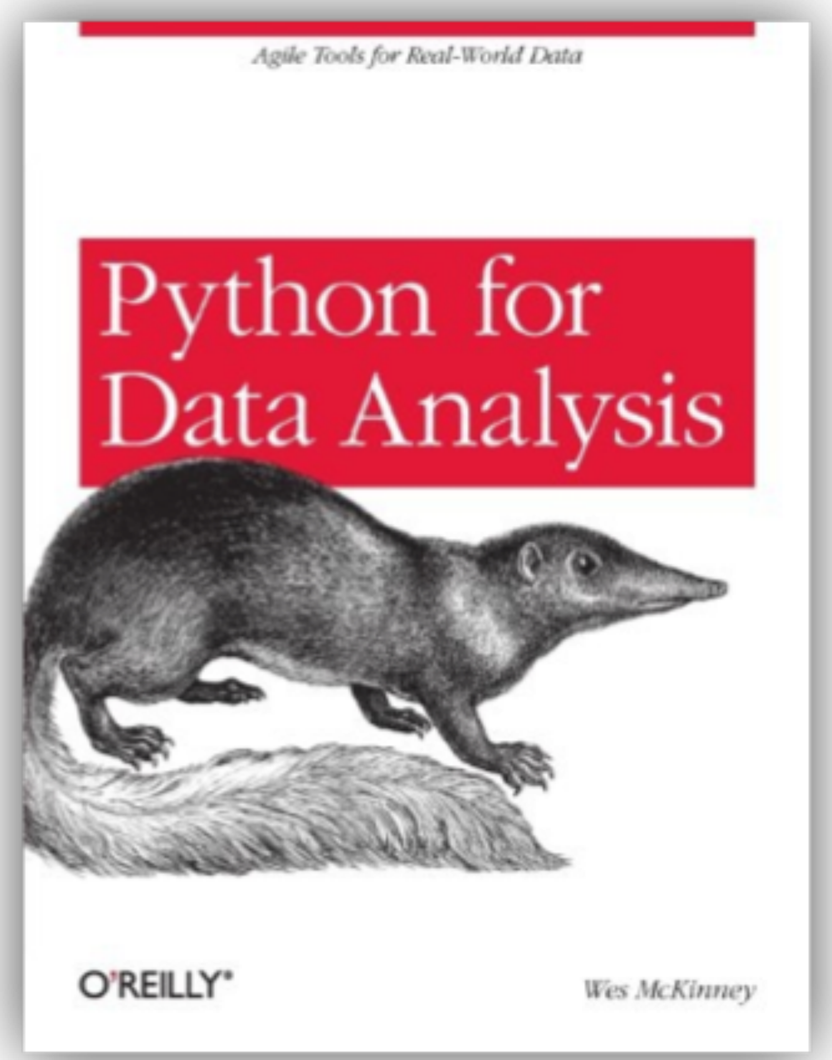
This talk

- Some commentary / notes on all the data frame interfaces out there
- Community / collaboration challenges
- Opinions

Disclaimer: This is a nuanced discussion

Who am I?

- Originator of pandas (2008 -)
- Financial analytics in R / Python starting 2007
- 2010-2012
  - Hiatus from gainful employment
  - Make pandas ready for primetime
  - Write "Python for Data Analysis”
- 2013-2014: DataPad with Chang She & co
- 2014 - : Open source development @ Cloudera



Data frame, in brief

- A table-like data structure
- An API / user interface for the table
  - Selecting data
  - Math and relational algebra (join, filter, etc.)
  - File / database IO
  - *ad infinitum*

cloudera

Some axes of comparison

- Data structure internals (types, in-memory representation, etc.)
- Basic table API
- Relational algebra support
- Group-by / split-apply-combine API
- Performance, memory use, evaluation semantics
- Missing data
- Data tidying / ETL tools
- IO utilities
- Domain specific tools (e.g. time series)
- …

cloudera

The Great Data Tool Decoupling™

- Thesis: over time, user interfaces, data storage, and execution engines will decouple and specialize
- In fact, you should really want this to happen
  - Share systems among languages
  - Reduce fragmentation and "lock-in"
  - Shift developer focus to usability
- Prediction: we'll be there by 2025; sooner if we all get our act together

cloudera

Crafting quality data tools

- Quality / usefulness is usually forged by the fire of battle
- Real world use cases and social proof trump theory
- Eat that dog food
- When in doubt? Look at the test suite.

cloudera

Data frames, circa 2015

- Tight coupling amongst
  - User API
  - In-memory data representation
  - Serialization/deserialization tools
  - Analytics / computation
  - Code evaluation semantics
- Code sharing amongst languages fairly difficult, rarely happens in practice

In-memory representation a major problem

- Any algorithms written against a data frame implementation assume a particular custom
  - Memory layout
  - Type system (+ missing data representation)
  - Memory management strategy
- Due to organic growth of libraries / language ecosystems, there was never an effort to reach any design consensus
- Downstream symptoms: benchmark-driven development

Lies, damn lies, and benchmarks

- Usual targets of benchmarking:
  - IO (CSV and database reading)
  - Joins
  - Aggregation / group-by operations
- What’s actually being tested
  - Quality of algorithm implementation
  - Data representation
  - Memory access patterns / CPU cache efficiency

Example: group-by-aggregate

```
SELECT a, sum(b) AS total
FROM df
GROUP BY 1

df.groupby('a')
  .b.sum()

df %>% group_by('a')
   %>% summarise(total=sum(b))
```

What's actually being benchmarked

- CPU/IO efficiency for for scanning **a** and **b** columns
- Speed to push **a** values through a hash table (quality of hash table impl now an issue)
- Time to sum **b** values given known categorization (using hash table)
- Speed of creating result data frame

cloudera

Data types

- Primitive value types
  - Number (integer, floating point)
  - Boolean
  - String (UTF-8), Binary
  - Timestamp
- Complex / nested types
  - Lists
  - Structs
  - Maps

cloudera

Data types

Example Table Row

```
{
  'name': {
    'first': 'Wes',
    'last': 'McKinney'
  }
  'age': 30
  'numbers': [1, 2, 3, 4]
  'addresses': [
    {'city': 'New York',
     'state': 'NY'},
    {'city': 'San Francisco',
     'state': 'CA'},
  ]
  'keys': [('foo', 27), ('baz', 35)]
}
```

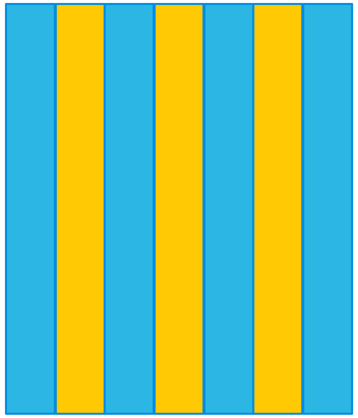
Table schema

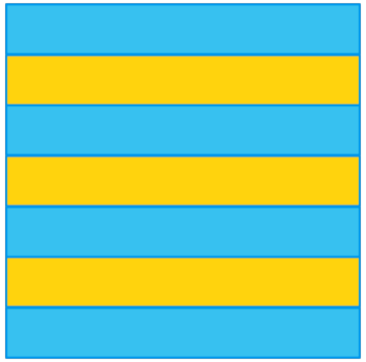
```
struct<
  name: struct<
    first: string,
    last: string
  >,
  age: int32,
  numbers: list<int32>,
  addresses: list<struct<
      city: string,
      state: string
    >>,
  keys: map<string, int>
>
```

cloudera

Table memory layout

- Columnar
  - Faster for analytics, single-column scans
  - Projections (column add/remove) cheap
  - Row appends harder
  - Operations across single rows are slower
- Row-oriented
  - Slower single-column scans
  - Projections expensive
  - Row appends easier
  - Operations across single rows are faster





cloudera

Serialization and protocols

- Text-based formats
  - CSV/TSV, JSON
- Binary formats
  - Avro
  - Thrift
  - Protocol buffers
  - Parquet (related: ColumnIO and RecordIO at Google)
  - ORCFile
  - HDF5
  - Language specific: NumPy, bcolz, Rdata, etc.

cloudera

Lossy / text-based formats

- Like CSV, JSON, etc.
- Can be read “easily” by anyone
- Expensive to read (parse) and write/generate
- Expensive to infer correct schema if not known
  - Still must validate parsed data, even if you know the schema
- Not a high fidelity format
  - Schema must be known
  - Data can be easily lost-in-translation
  - CSV cannot easily handle nested schemas

Binary data formats

- Some, like Parquet and ORCFile, are columnar / analytics-oriented
- Others (Avro, Thrift, Protobuf), designed for more general data transport and remote procedure calls (RPC) in distributed systems
- Language-specific formats generally don’t have full-featured reader-writers outside the primary language

Data representation across languages

- R: list of named R arrays (each containing data of a primitive R type)
  - Data frames have an implicit schema, but user does not interact with
  - Limited support for nested type values (JSON-like data)
- pandas: complex, but fundamentally data stored in NumPy arrays
  - No explicit table schema; NumPy complexities largely hidden from user
  - Limited / no built-in support for nested type values
- Missing / NULL values handled on a type-by-type basis (sometimes not at all)

Missing data

- R: uses special values to encode NA
- Python pandas: special values, but does not work for all types
- For arbitrary type data, best approach is to use a bit or byte to represent nullness/not-nullness

I am actively working to address infrastructural issues that have prevented collaboration / code sharing in tabular data tools

Some awesome R data frame stuff

- "Hadley Stack"
  - dplyr, tidyr
  - legacy: plyr, reshape2
  - ggplot2
- data.table (data.frame + indices, fast algorithms)
- xts : time series

cloudera

R data frames: rough edges

- Copy-on-write semantics
- API fragmentation / inconsistency
  - Use the “Hadley stack” for improved sanity
- Factor / String dichotomy
  - `stringsAsFactors=FALSE` a blessing and curse
- Somewhat limited type system

cloudera

dplyr

- Composable table API
- Good example of what the “decoupled” future might look like
  - New in-memory R/RCpp execution engine
  - SQL backends for large subset of API

Spark DataFrames

- R/pandas-inspired API for tabular data manipulation in Scala, Python, etc.
- Logical operation graphs rewritten internally in more efficient form
- Good interop with Spark SQL
- Some interoperability with pandas
- Partial API Decoupling!
- Low-level internals (DataFrame ~ RDD[Row]) are … not the best

pandas

- Several key data structures, data frame among them
- Considerably more complex internals than other data frame libraries
- Some good things
  - Born of need
  - A "batteries included" approach
  - Hierarchical axis labeling: addresses some hard use cases at expense of semantic complexity
  - Strong time series support

cloudera

pandas: rough edges

- Axis labelling can get in the way for folks needing “just a table”
- Ceded control of its type system / data rep’n from day 1 to NumPy
- Inefficient string handling (uses NumPy object arrays)
- Missing data handling less precise than other tools
- No C API; very difficult for external tools to interact with pandas objects

blaze

- Open source Python project run by Continuum Analytics
- Bespoke type system (“datashape”) supporting many kinds of nested schemas
- Data expression API with many execution backends (SQL, pandas, Spark, etc.)
- Does not have a “native” backend (was to be the—now aborted?—libdynd project)
- Another good example of the “decoupled” future

libdynd

- Next-gen NumPy-in-C++ project started by Continuum in 2011 to be a native backend for Blaze, appears not to be actively pursued
- Implements the new datashape protocol
- Standalone C++11/14 library, with deep Python binding
- Lots of interesting internals / implementation ideas

bcolz: the new carray / ctable

- Compressed in-memory / on disk columnar table structure
- Outgrowth of
  - carray (compressed-in-memory NumPy array)
  - blosc (multithreaded shuffling compression library)

Julia: DataFrames.jl

- Started by Harlan Harris & co
- Part of broader JuliaStats initiative
  - More R-like than pandas-like
  - Very active: > 50 contributors!
- Still comparatively early
  - Less comprehensive API
  - More limited IO capabilities

cloudera

Other data frames

- Saddle (Scala)
  - Dev'd by Adam Klein (ex-AQR) at Novus Partners (fintech startup)
  - Designed and used for financial use cases
- Deedle (F# / .NET)
  - Dev'd by AK's colleagues at BlueMountain (hedge fund)
- GraphLab / Dato
  - Really good C++ data frame with Python interface
  - Dual-licensed: AGPL + Commercial
- That's not all! Haskell, Go, etc…

We're not done yet

- The future is JSON-like
  - Support for nested types / semi-structured data is still weak
- Wanted: Apache-licensed, community standard C/C++ data frame data structure and analytics toolkit that we all use (R, Python, Julia)
- Bring on the Great Decoupling

cloudera

Thank you

@wesmckinn

