```
julia> using TimeSeries
```

Time and Data Analysis

In data analysis, a time dimension isn’t always necessary

Edgar Anderson recorded measurements on three species of irises but didn't include the time of day the observations were made

But that hasn’t stopped countless hours being spent analyzing his 150 timeless observations

Often though, the time dimension is an important piece of information

It provides a distance between observations

And many times human do interesting things on specific dates and times

TimeSeries is a lightweight package for data analysis when time matters

Package Design

parsimonious code base

```
[src (master)]
✈  git ls-files | xargs wc -l
       4 .timeseriesrc.jl
      23 TimeSeries.jl
     150 apply.jl
      74 combine.jl
      23 readwrite.jl
      58 split.jl
     203 timearray.jl
     535 total
```

TimeArray is an immutable type to work with time series data

```
timestamp::Vector{Date}
values::Array{T,N}
colnames::Vector{UTF8String}
```

timestamp length matches size of values rows

colnames length matches size of values column

dates cannot be duplicates

dates cannot be in a random unsorted order

The constructor also orders the rows starting with the first observation appearing first

API tour

```
julia> using MarketData

julia> ohlc
500x4 TimeArray{Float64,2} 2000-01-03 to 2001-12-31

              Open     High     Low      Close
2000-01-03 | 104.88   112.5    101.69   111.94
2000-01-04 | 108.25   110.62   101.19   102.5
2000-01-05 | 103.75   110.56   103.0    104.0
2000-01-06 | 106.12   107.0    95.0     95.0
⋮
2001-12-26 | 21.35    22.3     21.14    21.49
2001-12-27 | 21.58    22.25    21.58    22.07
2001-12-28 | 21.97    23.0     21.96    22.43
2001-12-31 | 22.51    22.66    21.83    21.9
```

Columns are indexed by string

```
julia> ohlc["Close"]
500x1 TimeArray{Float64,1} 2000-01-03 to 2001-12-31

             Close
2000-01-03 | 111.94
2000-01-04 | 102.5
2000-01-05 | 104.0
2000-01-06 | 95.0
⋮
2001-12-26 | 21.49
2001-12-27 | 22.07
2001-12-28 | 22.43
2001-12-31 | 21.9
```

Rows are indexed by date and date ranges

```
julia> ohlc[Date(2000,1,10)]
1x4 TimeArray{Float64,2} 2000-01-10 to 2000-01-10

             Open    High    Low     Close
2000-01-10 | 102.0   102.25  94.75   97.75


julia> ohlc[Date(2000,1,1):Date(2000,1,10)]
6x4 TimeArray{Float64,2} 2000-01-03 to 2000-01-10

             Open    High    Low     Close
2000-01-03 | 104.88  112.5   101.69  111.94
2000-01-04 | 108.25  110.62  101.19  102.5
2000-01-05 | 103.75  110.56  103.0   104.0
2000-01-06 | 106.12  107.0   95.0    95.0
2000-01-07 | 96.5    101.0   95.5    99.5
2000-01-10 | 102.0   102.25  94.75   97.75
```

Rows are indexed by integer and integer ranges

```
julia> ohlc[1]
1x4 TimeArray{Float64,2} 2000-01-03 to 2000-01-03

             Open      High      Low       Close
2000-01-03 | 104.88    112.5     101.69    111.94


julia> ohlc[1:3]
3x4 TimeArray{Float64,2} 2000-01-03 to 2000-01-05

             Open      High      Low       Close
2000-01-03 | 104.88    112.5     101.69    111.94
2000-01-04 | 108.25    110.62    101.19    102.5
2000-01-05 | 103.75    110.56    103.0     104.0
```

Columns and rows can be simultaneously indexed

```
julia> ohlc["Open", "Close"][[1:3,12]]
4x2 TimeArray{Float64,2} 2000-01-03 to 2000-01-19

             Open      Close
2000-01-03 | 104.88    111.94
2000-01-04 | 108.25    102.5
2000-01-05 | 103.75    104.0
2000-01-19 | 105.62    106.56
```

methods to subset between specific dates

```
julia> from(ohlc, 2001,12,27)
3x4 TimeArray{Float64,2} 2001-12-27 to 2001-12-31

             Open   High   Low    Close
2001-12-27 | 21.58  22.25  21.58  22.07
2001-12-28 | 21.97  23.0   21.96  22.43
2001-12-31 | 22.51  22.66  21.83  21.9


julia> to(ohlc, 2000,1,5)
3x4 TimeArray{Float64,2} 2000-01-03 to 2000-01-05

             Open    High    Low     Close
2000-01-03 | 104.88  112.5   101.69  111.94
2000-01-04 | 108.25  110.62  101.19  102.5
2000-01-05 | 103.75  110.56  103.0   104.0
```

aggregate on specific time period

```
julia> by(ohlc, 1, period=dayofweek)  # Mondays
95x4 TimeArray{Float64,2} 2000-01-03 to 2001-12-31

             Open    High    Low     Close
2000-01-03 | 104.88  112.5   101.69  111.94
2000-01-10 | 102.0   102.25  94.75   97.75
2000-01-24 | 108.44  112.75  105.12  106.25
2000-01-31 | 101.0   103.88  94.5    103.75
⋮
2001-12-10 | 22.29   22.99   22.23   22.54
2001-12-17 | 20.4    21.0    20.19   20.62
2001-12-24 | 20.9    21.45   20.9    21.36
2001-12-31 | 22.51   22.66   21.83   21.9
```

collapse allows control on what values are aggregated to larger time frame

```
julia> collapse(ohlc["Close"], last, period=month)
24x1 TimeArray{Float64,1} 2000-01-31 to 2001-12-31

             Close
2000-01-31 | 103.75
2000-02-29 | 114.62
2000-03-31 | 135.81
2000-04-28 | 124.06
⋮
2001-09-28 | 15.51
2001-10-31 | 17.56
2001-11-30 | 21.3
2001-12-31 | 21.9
```

element-wise operations

```
julia> ohlc["High"] .- ohlc["Close"]
500x1 TimeArray{Float64,1} 2000-01-03 to 2001-12-31

             Hi.-Cl
2000-01-03 | 0.56
2000-01-04 | 8.12
2000-01-05 | 6.56
2000-01-06 | 12.0
⋮
2001-12-26 | 0.81
2001-12-27 | 0.18
2001-12-28 | 0.57
2001-12-31 | 0.76
```

```
julia> ohlc["Open"] .> ohlc["Close"]
500x1 TimeArray{Bool,1} 2000-01-03 to 2001-12-31

             Op.>Cl
2000-01-03 | false
2000-01-04 | true
2000-01-05 | false
2000-01-06 | true
⋮
2001-12-26 | false
2001-12-27 | false
2001-12-28 | false
2001-12-31 | true
```

find when a condition is met

```
julia> greendays = findwhen(ohlc["Close"] .> ohlc["Open"]);

julia> typeof(greendays)
Array{Date,1}

julia> ohlc[greendays]
244x4 TimeArray{Float64,2} 2000-01-03 to 2001-12-28

             Open    High    Low     Close
2000-01-03 | 104.88  112.5   101.69  111.94
2000-01-05 | 103.75  110.56  103.0   104.0
2000-01-07 | 96.5    101.0   95.5    99.5
2000-01-13 | 94.48   98.75   92.5    96.75
⋮
2001-12-24 | 20.9    21.45   20.9    21.36
2001-12-26 | 21.35   22.3    21.14   21.49
2001-12-27 | 21.58   22.25   21.58   22.07
2001-12-28 | 21.97   23.0    21.96   22.43
```

```
julia> reddays = findall(ohlc["Close"] .< ohlc["Open"]);

julia> typeof(reddays)
Array{Int64,1}

julia> ohlc[reddays]
252x4 TimeArray{Float64,2} 2000-01-04 to 2001-12-31

             Open     High     Low      Close
2000-01-04 | 108.25   110.62   101.19   102.5
2000-01-06 | 106.12   107.0    95.0     95.0
2000-01-10 | 102.0    102.25   94.75    97.75
2000-01-11 | 95.94    99.38    90.5     92.75
⋮
2001-12-14 | 20.73    20.83    20.09    20.39
2001-12-20 | 21.4     21.47    20.62    20.67
2001-12-21 | 21.01    21.54    20.8     21.0
2001-12-31 | 22.51    22.66    21.83    21.9
```

time-related transformations

```
julia> lag(ohlc["Open"])
499x1 TimeArray{Float64,1} 2000-01-04 to 2001-12-31

             Open
2000-01-04 | 104.88
2000-01-05 | 108.25
2000-01-06 | 103.75
2000-01-07 | 106.12
⋮
2001-12-26 | 20.9
2001-12-27 | 21.35
2001-12-28 | 21.58
2001-12-31 | 21.97
```

```
julia> percentchange(ohlc["Close"], method="log")
499x1 TimeArray{Float64,1} 2000-01-04 to 2001-12-31

              Close
2000-01-04 | -0.09
2000-01-05 | 0.01
2000-01-06 | -0.09
2000-01-07 | 0.05
⋮
2001-12-26 | 0.01
2001-12-27 | 0.03
2001-12-28 | 0.02
2001-12-31 | -0.02
```

```
julia> moving(ohlc["Close"], mean, 20)
481x1 TimeArray{Float64,1} 2000-01-31 to 2001-12-31

             Close
2000-01-31 | 103.36
2000-02-01 | 102.78
2000-02-02 | 102.59
2000-02-03 | 102.56
⋮
2001-12-26 | 21.49
2001-12-27 | 21.56
2001-12-28 | 21.67
2001-12-31 | 21.7
```

```
julia> upto(ohlc["Close"], sum)
500x1 TimeArray{Float64,1} 2000-01-03 to 2001-12-31

             Close
2000-01-03 | 111.94
2000-01-04 | 214.44
2000-01-05 | 318.44
2000-01-06 | 413.44
⋮
2001-12-26 | 23028.84
2001-12-27 | 23050.91
2001-12-28 | 23073.34
2001-12-31 | 23095.24
```

basecall uses fast base methods

```
julia> BA["Close"]
13090x1 TimeArray{Float64,1} 1962-01-02 to 2013-12-31

              Close
1962-01-02 | 50.0
1962-01-03 | 51.0
1962-01-04 | 50.5
1962-01-05 | 49.5
⋮
2013-12-26 | 138.27
2013-12-27 | 136.9
2013-12-30 | 135.92
2013-12-31 | 136.49


julia> @time upto(BA["Close"], sum);
elapsed time: 0.092170981 seconds (4663992 bytes allocated)

julia> @time basecall(BA["Close"], cumsum);
elapsed time: 0.0099391 seconds (3990200 bytes allocated)
```

merging two TimeArrays

```
julia> @time merge(BA["High"], CAT["Low"])
elapsed time: 1.776906707 seconds (12571880 bytes allocated)
13090x2 TimeArray{Float64,2} 1962-01-02 to 2013-12-31

             High     Low
1962-01-02 | 50.88    38.12
1962-01-03 | 51.75    38.12
1962-01-04 | 51.88    39.75
1962-01-05 | 50.75    39.75
⋮
2013-12-26 | 138.59   90.7
2013-12-27 | 138.88   90.56
2013-12-30 | 137.37   90.28
2013-12-31 | 137.05   90.46
```

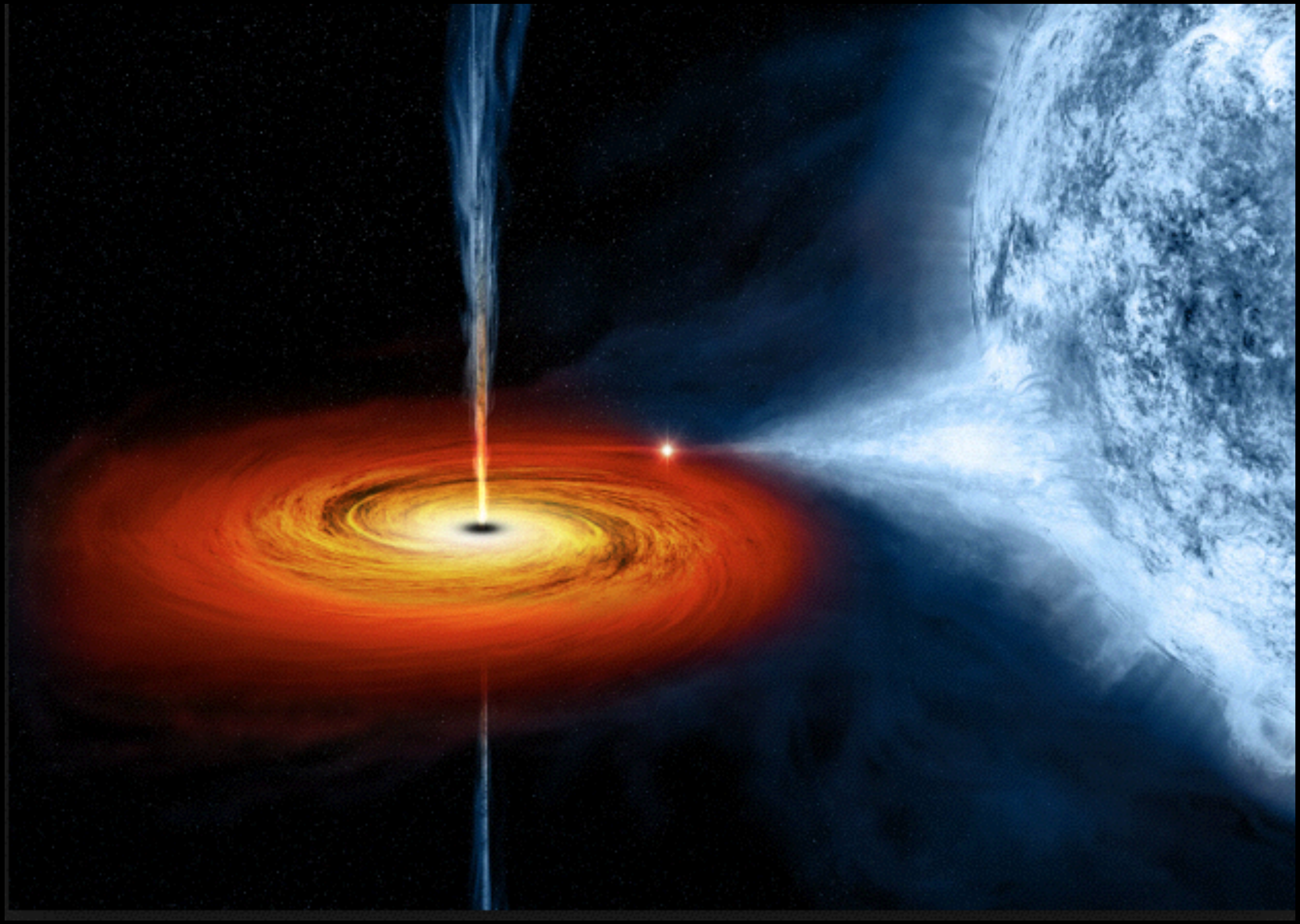
slow
column names are not unique
only inner joins are supported

pull requests are welcome!

When Time Disappears

artifact dates are not preserved

This is what happens when your dates are consumed by computation or shifting



```
julia> ohlc[1]
1x4 TimeArray{Float64,2} 2000-01-03 to 2000-01-03

             Open    High    Low     Close
2000-01-03 | 104.88  112.5   101.69  111.94


julia> lag(ohlc)
499x4 TimeArray{Float64,2} 2000-01-04 to 2001-12-31

             Open    High    Low     Close
2000-01-04 | 104.88  112.5   101.69  111.94
2000-01-05 | 108.25  110.62  101.19  102.5
2000-01-06 | 103.75  110.56  103.0   104.0
2000-01-07 | 106.12  107.0   95.0    95.0
⋮
2001-12-26 | 20.9    21.45   20.9    21.36
2001-12-27 | 21.35   22.3    21.14   21.49
2001-12-28 | 21.58   22.25   21.58   22.07
2001-12-31 | 21.97   23.0    21.96   22.43
```

Rather than conflate NaN with missingness
or implement the DataFrames NA type,
TimeSeries tosses consumed values
into black holes

Some packages that use TimeSeries don't have this luxury and have introduced NaN placeholders similar to how pandas handles this

```
julia> using Quandl

julia> quandl("CHRIS/CME_DK1") # Class IV Milk Futures
100x8 TimeArray{Float64,2} 2014-01-14 to 2014-06-05

             Open    High    Low     Last   Change  Settle  Volume  Open Interest
2014-01-14 | NaN     22.05   22.02   NaN    NaN     22.05   NaN     NaN
2014-01-15 | NaN     22.1    21.99   NaN    NaN     22.09   NaN     NaN
2014-01-16 | NaN     22.2    22.1    NaN    NaN     22.1    NaN     NaN
2014-01-17 | NaN     22.2    22.17   NaN    NaN     22.2    NaN     NaN
⋮
2014-06-02 | NaN     NaN     NaN     NaN    NaN     22.59   0.0     1660.0
2014-06-03 | NaN     NaN     22.54   NaN    0.03    22.56   0.0     1660.0
2014-06-04 | 22.55   22.65   22.55   NaN    NaN     22.55   11.0    1565.0
2014-06-05 | 22.57   22.7    22.57   NaN    0.14    22.69   6.0     1567.0
```

Time Series in R and Python

R’s xts and Python’s pandas
for comparison

R code to duplicate data object

```
> library(quantmod)
> getSymbols('AAPL', from='2000-1-1', to='2001-12-31')
[1] "AAPL"
> head(AAPL)
           AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted
2000-01-03    104.88    112.50   101.69     111.94   133949200          3.82
2000-01-04    108.25    110.62   101.19     102.50   128094400          3.50
2000-01-05    103.75    110.56   103.00     104.00   194580400          3.55
2000-01-06    106.13    107.00    95.00      95.00   191993200          3.24
2000-01-07     96.50    101.00    95.50      99.50   115183600          3.40
2000-01-10    102.00    102.25    94.75      97.75   126266000          3.34
```

pandas code to duplicate data object

```
In [1]: from pandas import *

In [2]: from pandas.io.data import DataReader

In [3]: from datetime import datetime

In [4]: AAPL = DataReader("AAPL", "yahoo", datetime(2000,1,1), datetime(2001,12,31))

In [5]: AAPL[0:3]
Out[5]:
              Open    High     Low   Close     Volume  Adj Close
Date
2000-01-03  104.88  112.50  101.69  111.94  133949200       3.82
2000-01-04  108.25  110.62  101.19  102.50  128094400       3.50
2000-01-05  103.75  110.56  103.00  104.00  194580400       3.55
```

Both xts and pandas support date strings for subsetting data

This is a convenience that TimeSeries
is taking a pass on

R’s xts doesn’t do error-checking that your
string is ISO-compliant and fails
silently when it isn’t

Both xts and pandas preserve dates (rows) of data consumed by shifting or calculation

```
> lagged = lag(AAPL)
> lagged[1:3]
           AAPL.Open AAPL.High AAPL.Low AAPL.Close AAPL.Volume AAPL.Adjusted
2000-01-03        NA        NA       NA         NA          NA            NA
2000-01-04    104.88    112.50   101.69     111.94   133949200          3.82
2000-01-05    108.25    110.62   101.19     102.50   128094400          3.50
```

```
In [5]: lag = AAPL.shift(1)

In [6]: lag[0:3]
Out[6]:
              Open    High     Low   Close     Volume  Adj Close
Date
2000-01-03     NaN     NaN     NaN     NaN        NaN        NaN
2000-01-04  104.88  112.50  101.69  111.94  133949200       3.82
2000-01-05  108.25  110.62  101.19  102.50  128094400       3.50
```

Both xts and pandas show floats disguised as integers, when they were converted to floats from integers to fit into their array

TimeSeries defaults to showing
all elements of a float array
as floats, but you can modify that

You can also modify how many decimals are displayed

.timeseriesrc.jl

```julia
###### customizable show ########

const DECIMALS = 4   # default value is 2
const SHOWINT = true # defaults to false
```

```
julia> ohlcv[1:3]
3x5 TimeArray{Float64,2} 2000-01-03 to 2000-01-05

             Open      High      Low       Close     Volume
2000-01-03 | 104.88    112.5     101.69    111.94    4783900
2000-01-04 | 108.25    110.62    101.19    102.5     4574800
2000-01-05 | 103.75    110.56    103.0     104.0     6949300


julia> percentchange(cl)[1:3]
3x1 TimeArray{Float64,2} 2000-01-04 to 2000-01-06

             Close
2000-01-04 | -0.0843
2000-01-05 | 0.0146
2000-01-06 | -0.0865
```

TimeSeries lives in the JuliaStats organization

Special thanks to Jacob Quinn for his work on the Dates.jl package. TimeSeries now depends on that package.

The End