pygdf Documentation

Release 0.1.0

Continuum Analytics

Jan 23, 2018

Contents:

CHAPTER 1

API Reference

1.1 DataFrame

class pygdf.dataframe.**DataFrame**(*name_series=None*)

A GPU Dataframe object.

Examples

Build dataframe with *__setitem__*

```
>>> from pygdf.dataframe import DataFrame
>>> df = DataFrame()
>>> df['key'] = [0, 1, 2, 3, 4]
>>> df['val'] = [float(i + 10) for i in range(5)]  # insert column
>>> df
  key val
0 0   10.0
1 1   11.0
2 2   12.0
3 3   13.0
4 4   14.0
>>> len(df)
5
```

Build dataframe with initializer

```
>>> import numpy as np
>>> df2 = DataFrame([('a', np.arange(10)),
...                  ('b', np.random.random(10))])
>>> df2
  a b
0 0 0.777831724018
1 1 0.604480034669
```

```
2 2 0.664111858618
3 3 0.887777513028
4 4 0.55838311246
[5 more rows]
```

Convert from a Pandas DataFrame.

```
>>> import pandas as pd
>>> from pygdf.dataframe import DataFrame
>>> pdf = pd.DataFrame({'a': [0, 1, 2, 3],
...                     'b': [0.1, 0.2, None, 0.3]})
>>> pdf
a    b
0  0  0.1
1  1  0.2
2  2  NaN
3  3  0.3
>>> df = DataFrame.from_pandas(pdf)
>>> df
a b
0 0 0.1
1 1 0.2
2 2 nan
3 3 0.3
```

Attributes

columns	Returns a tuple of columns
dtypes	Return the dtypes in this object.
index	Returns the index of the DataFrame
loc	Returns a label-based indexer for row-slicing and column selection.

Methods

add_column(name, data)	Add a column
apply_chunks(func, incols, outcols[, ...])	Transform user-specified chunks using the user-provided function.
apply_rows(func, incols, outcols, kwargs)	Transform each row using the user-provided function.
as_gpu_matrix([columns])	Convert to a matrix in device memory.
as_matrix([columns])	Covert to a matrix in host memory.
copy()	Shallow copy this dataframe
drop_column(name)	Drop a column by *name*
from_pandas(dataframe)	Convert from a Pandas DataFrame.
from_records(data[, index, columns])	Convert from a numpy recarray or structured array.
groupby(by[, sort, as_index])	Groupby
head([n])	
join(other[, on, how, lsuffix, rsuffix, sort])	Join columns with other DataFrame on index or on a key column.
label_encoding(column, prefix, cats[, ...])	Encode labels in a column with label encoding.

Continued on next page

Table 1.2 – continued from previous page

nlargest(n, columns[, keep])	Get the rows of the DataFrame sorted by the n largest value of *columns*
nsmallest(n, columns[, keep])	Get the rows of the DataFrame sorted by the n smallest value of *columns*
one_hot_encoding(column, prefix, cats[, ...])	Expand a column with one-hot-encoding.
query(expr)	Query with a boolean expression using Numba to compile a GPU kernel.
reset_index()	
set_index(index)	Return a new DataFrame with a new index
sort_index([ascending])	Sort by the index
sort_values(by[, ascending])	Sort by values.
take(positions[, ignore_index])	
to_pandas()	Convert to a Pandas DataFrame.
to_records([index])	Covert to a numpy recarray
to_string([nrows, ncols])	Convert to string

add_column (*name*, *data*)
Add a column

Parameters **name** : str

Name of column to be added.

data : Series, array-like

Values to be added.

apply_chunks (*func*, *incols*, *outcols*, *kwargs={}*, *chunks=None*, *tpb=1*)
Transform user-specified chunks using the user-provided function.

Parameters **func** : function

The transformation function that will be executed on the CUDA GPU.

incols: list

A list of names of input columns.

outcols: dict

A dictionary of output column names and their dtype.

kwargs: dict

name-value of extra arguments. These values are passed directly into the function.

chunks : int or Series-like

If it is an `int`, it is the chunksize. If it is an array, it contains integer offset for the start of each chunk. The span of a chunk for chunk i-th is `data[chunks[i] : chunks[i + 1]]` for any `i + 1 < chunks.size`; or, `data[chunks[i]:]` for the `i == len(chunks) - 1`.

tpb : int; optional

It is the thread-per-block for the underlying kernel. The default uses 1 thread to emulate serial execution for each chunk. It is a good starting point but inefficient. Its maximum possible value is limited by the available CUDA GPU resources.

See also:

apply_rows

Examples

For `tpb > 1`, `func` is executed by `tpb` number of threads concurrently. To access the thread id and count, use `numba.cuda.threadIdx.x` and `numba.cuda.blockDim.x`, respectively (See numba CUDA kernel documentation).

In the example below, the *kernel* is invoked concurrently on each specified chunk. The *kernel* computes the corresponding output for the chunk. By looping over the range `range(cuda.threadIdx.x, in1.size, cuda.blockDim.x)`, the *kernel* function can be used with any *tpb* in a efficient manner.

```
>>> from numba import cuda
>>> def kernel(in1, in2, in3, out1):
...     for i in range(cuda.threadIdx.x, in1.size, cuda.blockDim.x):
...         x = in1[i]
...         y = in2[i]
...         z = in3[i]
...         out1[i] = x * y + z
```

apply_rows (*func*, *incols*, *outcols*, *kwargs*)

Transform each row using the user-provided function.

Parameters **func** : function

The transformation function that will be executed on the CUDA GPU.

incols: list

A list of names of input columns.

outcols: dict

A dictionary of output column names and their dtype.

kwargs: dict

name-value of extra arguments. These values are passed directly into the function.

Examples

With a `DataFrame` like so:

```
>>> df = DataFrame()
>>> df['in1'] = in1 = np.arange(nelem)
>>> df['in2'] = in2 = np.arange(nelem)
>>> df['in3'] = in3 = np.arange(nelem)
```

Define the user function for `.apply_rows`:

```
>>> def kernel(in1, in2, in3, out1, out2, extra1, extra2):
...     for i, (x, y, z) in enumerate(zip(in1, in2, in3)):
...         out1[i] = extra2 * x - extra1 * y
...         out2[i] = y - extra1 * z
```

The user function should loop over the columns and set the output for each row. Each iteration of the loop **MUST** be independent of each other. The order of the loop execution can be arbitrary.

Call `.apply_rows` with the name of the input columns, the name and dtype of the output columns, and, optionally, a dict of extra arguments.

```
>>> outdf = df.apply_rows(kernel,
...                       incols=['in1', 'in2', 'in3'],
...                       outcols=dict(out1=np.float64,
...                                    out2=np.float64),
...                       kwargs=dict(extra1=2.3, extra2=3.4))
```

Notes

When `func` is invoked, the array args corresponding to the input/output are strided in a way that improves parallelism on the GPU. The loop in the function may look like serial code but it will be executed concurrently by multiple threads.

as_gpu_matrix(*columns=None*)
Convert to a matrix in device memory.

Parameters **columns: sequence of str**

List of a column names to be extracted. The order is preserved. If None is specified, all columns are used.

Returns A (nrow x ncol) numpy ndarray in "F" order.

as_matrix(*columns=None*)
Covert to a matrix in host memory.

Parameters **columns: sequence of str**

List of a column names to be extracted. The order is preserved. If None is specified, all columns are used.

Returns A (nrow x ncol) numpy ndarray in "F" order.

columns
Returns a tuple of columns

copy()
Shallow copy this dataframe

drop_column(*name*)
Drop a column by *name*

dtypes
Return the dtypes in this object.

classmethod from_pandas(*dataframe*)
Convert from a Pandas DataFrame.

Raises **TypeError for invalid input type.**

classmethod from_records(*data*, *index=None*, *columns=None*)
Convert from a numpy recarray or structured array.

Parameters **data** : numpy structured dtype or recarray

index : str

The name of the index column in *data*. If None, the default index is used.

columns : list of str

List of column names to include.

Returns DataFrame

groupby(*by*, *sort=False*, *as_index=False*)
Groupby

Parameters **by** : list-of-str or str

Column name(s) to form that groups by.

sort : bool

Force sorting group keys. Depends on the underlying algorithm. Current algorithm always sort.

as_index : bool; defaults to False

Must be False. Provided to be API compatible with pandas. The keys are always left as regular columns in the result.

Returns The groupby object

Notes

Unlike pandas, this groupby operation behaves like a SQL groupby. No empty rows are returned. (For categorical keys, pandas returns rows for all categories even if they are no corresponding values.)

Only a minimal number of operations is implemented so far.

- Only *by* argument is supported.
- The output is always sorted according to the *by* columns.
- Since we don't support multiindex, the *by* columns are stored as regular columns.

index
Returns the index of the DataFrame

join(*other*, *on=None*, *how='left'*, *lsuffix=''*, *rsuffix=''*, *sort=False*)
Join columns with other DataFrame on index or on a key column.

Parameters **other** : DataFrame

how : str

Only accepts "left", "right", "inner", "outer"

lsuffix, rsuffix : str

The suffices to add to the left (*lsuffix*) and right (*rsuffix*) column names when avoiding conflicts.

sort : bool

Set to True to ensure sorted ordering.

Returns joined : DataFrame

Notes

Difference from pandas:

- *other* must be a single DataFrame for now.
- *on* is not supported yet due to lack of multi-index support.

label_encoding(*column*, *prefix*, *cats*, *prefix_sep='_'*, *dtype=None*, *na_sentinel=-1*)
Encode labels in a column with label encoding.

Parameters **column** : str

the source column with binary encoding for the data.

prefix : str

the new column name prefix.

cats : sequence of ints

the sequence of categories as integers.

prefix_sep : str

the separator between the prefix and the category.

dtype :

the dtype for the outputs; see Series.label_encoding

na_sentinel : number

Value to indicate missing category.

Returns

——-

a new dataframe with a new column append for the coded values.

loc
Returns a label-based indexer for row-slicing and column selection.

Examples

```
>>> df = DataFrame([('a', list(range(20))),
...                 ('b', list(range(20))),
...                 ('c', list(range(20)))])
# get rows from index 2 to index 5 from 'a' and 'b' columns.
>>> df.loc[2:5, ['a', 'b']]
     a    b
2    2    2
3    3    3
4    4    4
5    5    5
```

nlargest(*n*, *columns*, *keep='first'*)
Get the rows of the DataFrame sorted by the n largest value of *columns*

Difference from pandas: * Only a single column is supported in *columns*

nsmallest(*n*, *columns*, *keep='first'*)
Get the rows of the DataFrame sorted by the n smallest value of *columns*

Difference from pandas: * Only a single column is supported in *columns*

one_hot_encoding(*column*, *prefix*, *cats*, *prefix_sep='_'*, *dtype='float64'*)
Expand a column with one-hot-encoding.

Parameters **column** : str

the source column with binary encoding for the data.

prefix : str

the new column name prefix.

cats : sequence of ints

the sequence of categories as integers.

prefix_sep : str

the separator between the prefix and the category.

dtype :

the dtype for the outputs; defaults to float64.

Returns a new dataframe with new columns append for each category.

query(*expr*)

Query with a boolean expression using Numba to compile a GPU kernel.

See pandas.DataFrame.query.

Parameters **expr** : str

A boolean expression. Names in the expression refers to the columns. Any name prefixed with @ refer to the variables in the calling environment.

Returns **filtered** : DataFrame

set_index(*index*)

Return a new DataFrame with a new index

Parameters **index** : Index, Series-convertible or str

Index: the new index. Series-convertible: values for the new index. str: name of column to be used as series

sort_index(*ascending=True*)

Sort by the index

sort_values(*by*, *ascending=True*)

Sort by values.

Difference from pandas: * *by* must be the name of a single column. * Support axis='index' only. * Not supporting: inplace, kind, na_position

Details: Uses parallel radixsort, which is a stable sort.

to_pandas()

Convert to a Pandas DataFrame.

to_records(*index=True*)

Covert to a numpy recarray

Parameters **index** : bool

Whether to include the index in the output.

Returns numpy recarray

to_string(*nrows=NOTSET*, *ncols=NOTSET*)

Convert to string

Parameters **nrows** : int

Maximum number of rows to show. If it is None, all rows are shown.

ncols : int

Maximum number of columns to show. If it is None, all columns are shown.

1.2 Series

class pygdf.dataframe.**Series**(*data*, *index=None*)

Data and null-masks.

`Series` objects are used as columns of `DataFrame`.

Attributes

cat	
data	The gpu buffer for the data
dtype	dtype of the Series
has_null_mask	A boolean indicating whether a null-mask is needed
index	The index object
null_count	Number of null values
nullmask	The gpu buffer for the null-mask
valid_count	Number of non-null values

Methods

append(arbitrary)	Append values from another `Series` or array-like object.
applymap(udf[, out_dtype])	Apply a elemenwise function to transform the values in the Column.
argsort([ascending])	Returns a Series of int64 index that will sort the series.
as_index()	
as_mask()	Convert booleans to bitmask
astype(dtype)	Convert to the given `dtype`.
ceil()	Rounds each value upward to the smallest integral value not less than the original.
count()	The number of non-null values
factorize([na_sentinel])	Encode the input values as integer labels
fillna(value)	Fill null values with `value`.
find_first_value(value)	Returns offset of first value that matches
find_last_value(value)	Returns offset of last value that matches
floor()	Rounds each value downward to the largest integral value not greater than the original.
from_categorical(categorical[, codes])	Creates from a pandas.Categorical
from_masked_array(data, mask[, null_count])	Create a Series with null-mask.
head([n])	
label_encoding(cats[, dtype, na_sentinel])	Perform label encoding
max()	Compute the max of the series
mean()	Compute the mean of the series

Continued on next page

Table 1.4 – continued from previous page

mean_var([ddof])	Compute mean and variance at the same time.
min()	Compute the min of the series
nlargest([n, keep])	Returns a new Series of the *n* largest element.
nsmallest([n, keep])	Returns a new Series of the *n* smallest element.
one_hot_encoding(cats[, dtype])	Perform one-hot-encoding
reset_index()	Reset index to RangeIndex
reverse()	Reverse the Series
scale()	Scale values to [0, 1] in float64
set_index(index)	Returns a new Series with a different index.
set_mask(mask[, null_count])	Create new Series by setting a mask array.
sort_index([ascending])	Sort by the index.
sort_values([ascending])	Sort by values.
std([ddof])	Compute the standard deviation of the series
sum()	Compute the sum of the series
take(indices[, ignore_index])	Return Series by taking values from the corresponding *indices*.
to_array([fillna])	Get a dense numpy array for the data.
to_gpu_array([fillna])	Get a dense numba device array for the data.
to_pandas([index])	
to_string([nrows])	Convert to string
unique()	Returns unique values of this Series.
unique_k(k)	
values_to_string([nrows])	Returns a list of string for each element.
var([ddof])	Compute the variance of the series

append(*arbitrary*)
Append values from another `Series` or array-like object. Returns a new copy with the index resetted.

applymap(*udf*, *out_dtype=None*)
Apply a elemenwise function to transform the values in the Column.

The user function is expected to take one argument and return the result, which will be stored to the output Series. The function cannot reference globals except for other simple scalar objects.

Parameters **udf** : function

Wrapped by `numba.cuda.jit` for call on the GPU as a device function.

out_dtype : numpy.dtype; optional

The dtype for use in the output. By default, the result will have the same dtype as the source.

Returns **result** : Series

The mask and index are preserved.

argsort(*ascending=True*)
Returns a Series of int64 index that will sort the series.

Uses stable parallel radixsort.

Returns result: Series

as_mask()
Convert booleans to bitmask

Returns device array

astype(*dtype*)
Convert to the given `dtype`.

Returns If the dtype changed, a new `Series` is returned by casting each

values to the given dtype.

If the dtype is not changed, `self` is returned.

ceil()
Rounds each value upward to the smallest integral value not less than the original.

Returns a new Series.

count()
The number of non-null values

data
The gpu buffer for the data

dtype
dtype of the Series

factorize(*na_sentinel=-1*)
Encode the input values as integer labels

Parameters **na_sentinel** : number

Value to indicate missing category.

Returns **(labels, cats)** : (Series, Series)

- *labels* contains the encoded values
- *cats* contains the categories in order that the N-th item corresponds to the (N-1) code.

fillna(*value*)
Fill null values with `value`.

Returns a copy with null filled.

find_first_value(*value*)
Returns offset of first value that matches

find_last_value(*value*)
Returns offset of last value that matches

floor()
Rounds each value downward to the largest integral value not greater than the original.

Returns a new Series.

classmethod from_categorical(*categorical*, *codes=None*)
Creates from a pandas.Categorical

If `codes` is defined, use it instead of `categorical.codes`

classmethod from_masked_array(*data*, *mask*, *null_count=None*)
Create a Series with null-mask. This is equivalent to:

Series(data).set_mask(mask, null_count=null_count)

Parameters **data** : 1D array-like

The values. Null values must not be skipped. They can appear as garbage values.

mask : 1D array-like of numpy.uint8

The null-mask. Valid values are marked as `1`; otherwise `0`. The mask bit given the data index `idx` is computed as:

```
(mask[idx // 8] >> (idx % 8)) & 1
```

null_count : int, optional

The number of null values. If None, it is calculated automatically.

has_null_mask
A boolean indicating whether a null-mask is needed

index
The index object

label_encoding(*cats*, *dtype=None*, *na_sentinel=-1*)
Perform label encoding

Parameters **values** : sequence of input values

dtype: numpy.dtype; optional

Specifies the output dtype. If *None* is given, the smallest possible integer dtype (starting with np.int32) is used.

na_sentinel : number

Value to indicate missing category.

Returns

———-

A sequence of encoded labels with value between 0 and n-1 classes(cats)

max()
Compute the max of the series

mean()
Compute the mean of the series

mean_var(*ddof=1*)
Compute mean and variance at the same time.

min()
Compute the min of the series

nlargest(*n=5*, *keep='first'*)
Returns a new Series of the *n* largest element.

nsmallest(*n=5*, *keep='first'*)
Returns a new Series of the *n* smallest element.

null_count
Number of null values

nullmask
The gpu buffer for the null-mask

one_hot_encoding(*cats*, *dtype='float64'*)
Perform one-hot-encoding

Parameters **cats** : sequence of values

values representing each category.

dtype : numpy.dtype

specifies the output dtype.

Returns A sequence of new series for each category. Its length is determined

by the length of `cats`.

reset_index()
Reset index to RangeIndex

reverse()
Reverse the Series

scale()
Scale values to [0, 1] in float64

set_index(*index*)
Returns a new Series with a different index.

Parameters **index** : Index, Series-convertible

the new index or values for the new index

set_mask(*mask*, *null_count=None*)
Create new Series by setting a mask array.

This will override the existing mask. The returned Series will reference the same data buffer as this Series.

Parameters **mask** : 1D array-like of numpy.uint8

The null-mask. Valid values are marked as `1`; otherwise `0`. The mask bit given the data index `idx` is computed as:

```
(mask[idx // 8] >> (idx % 8)) & 1
```

null_count : int, optional

The number of null values. If None, it is calculated automatically.

sort_index(*ascending=True*)
Sort by the index.

sort_values(*ascending=True*)
Sort by values.

Difference from pandas: * Support axis='index' only. * Not supporting: inplace, kind, na_position

Details: Uses parallel radixsort, which is a stable sort.

std(*ddof=1*)
Compute the standard deviation of the series

sum()
Compute the sum of the series

take(*indices*, *ignore_index=False*)
Return Series by taking values from the corresponding *indices*.

to_array(*fillna=None*)
Get a dense numpy array for the data.

Parameters **fillna** : str or None

Defaults to None, which will skip null values. If it equals "pandas", null values are filled with NaNs. Non integral dtype is promoted to np.float64.

Notes

if `fillna` is `None`, null values are skipped. Therefore, the output size could be smaller.

to_gpu_array(*fillna=None*)
Get a dense numba device array for the data.

Parameters **fillna** : str or None

See *fillna* in `.to_array`.

Notes

if `fillna` is `None`, null values are skipped. Therefore, the output size could be smaller.

to_string(*nrows=NOTSET*)
Convert to string

Parameters **nrows** : int

Maximum number of rows to show. If it is None, all rows are shown.

unique()
Returns unique values of this Series.

valid_count
Number of non-null values

values_to_string(*nrows=None*)
Returns a list of string for each element.

var(*ddof=1*)
Compute the variance of the series

1.3 Groupby

class pygdf.groupby.**Groupby**(*df*, *by*)
Groupby object returned by pygdf.DataFrame.groupby().

Methods

agg(args)	Invoke aggregation functions on the groups.
apply(function)	Apply a transformation function over the grouped chunk.
apply_grouped(function, **kwargs)	
as_df()	Get the intermediate dataframe after shuffling the rows into groups.
count()	Compute the count of each group
max()	Compute the max of each group
mean()	Compute the mean of each group
min()	Compute the min of each group
std()	Compute the std of each group

agg(*args*)
Invoke aggregation functions on the groups.

Parameters **args: dict, list, str, callable**

- **str** The aggregate function name.
- **callable** The aggregate function.
- **list** List of *str* or *callable* of the aggregate function.
- **dict** key-value pairs of source column name and list of aggregate functions as *str* or *callable*.

Returns **result** : DataFrame

apply(*function*)
Apply a transformation function over the grouped chunk.

as_df()
Get the intermediate dataframe after shuffling the rows into groups.

Returns **(df, segs)** : namedtuple

- df : DataFrame
- **segs** [Series] Beginning offsets of each group.

count()
Compute the count of each group

Returns **result** : DataFrame

max()
Compute the max of each group

Returns **result** : DataFrame

mean()
Compute the mean of each group

Returns **result** : DataFrame

min()
Compute the min of each group

Returns **result** : DataFrame

std()
Compute the std of each group

Returns **result** : DataFrame

1.4 GpuArrowReader

class pygdf.gpuarrow.**GpuArrowReader**(*schema_data*, *gpu_data*)

Methods

to_dict()	Return a dictionary of Series object

to_dict()
Return a dictionary of Series object

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Index