fletcher

Fletcher contributors

Jan 17, 2021

CONTENTS:

1 Motivation 1

2 Frequently Asked Questions 3

3 fletcher 5

4 Indices and tables 39

Python Module Index 41

Index 43

CHAPTER

ONE

MOTIVATION

Fletcher was started as weekend project to see how far you could get by using pandas' new `ExtensionArray` interface together with Numba and Apache Arrow to build a fast string type. Since then it has evolved into a project to explore the general issues that you experience when using a non-numpy storage for pandas columns.

Fletchers main aim is now to provide a general `ExtensionArray` implementation to support Apache Arrow-backed columns in `pandas`. We restrict ourselves in the development of fletcher to use only pure Python code and do all compilation just-in-time using Numba. On the one side, this makes distribution of this package much simpler as we don't ship native code, on the other side it is also dogfooding for the Apache Arrow developers working on this project to better understand what is needed to accelerate Python code using Numba on top of Apache Arrow.

While Fletcher is currently not in a state where you can use it in day-to-day work, it already provides valuable feedback to pandas on where its ExtensionArray interface is still bound to NumPy. In addition, it also provides feedback to Apache Arrow on what functionality is missing to use it as the backend of a DataFrame library. A long-term goal of fletcher is to provide enough input to the pandas and Apache Arrow community to eventually let pandas use `pyarrow` as a hard dependency for backing some of its column types.

As an end-user, you watch Fletcher's development over the next months when you are interested in the efficient implementation of the following data types:

- Nullability for numerical data (all other data supported by fletcher also supports efficient nullability masks)
- strings
- nested structures like `List[...]` or `Struct[...]` or any recursive combination of that.

CHAPTER

TWO

FREQUENTLY ASKED QUESTIONS

2.1 Roadmap

2.1.1 Should this be merged into `pandas` one day?

Yes, we definitely want to have parts of `fletcher` as part of `pandas`. There are plans for a native string type and a list type where Apache Arrow would be the preferable data structure.

2.1.2 If this is merged into `pandas`, is there still a need for `fletcher`?

Definitely! `fletcher` functions as glue for all available types in Apache Arrow as `pandas.ExtensionDtype`. Only a subset of these types will make its way into `pandas`, thus the string functions that return a boolean result in `pandas` will return a `pandas.BooleanArray` (bytemask-backed `numpy` array) whereas the ones in `fletcher` will keep returning a *`fletcher.FletcherBaseArray`* (bitmask-backed Arrow array).

Furthermore, an additional goal of `fletcher` is to support working with `numba` on top of `pyarrow.Array` structures. These utilities will not make their way into `pandas` as there we don't wont to introduce a hard `numba` dependency.

2.2 Development

CHAPTER

THREE

FLETCHER

3.1 fletcher package

3.1.1 Subpackages

fletcher.algorithms package

Subpackages

fletcher.algorithms.utils package

Submodules

fletcher.algorithms.utils.chunking module

Utility functions to deal with chunked arrays.

fletcher.algorithms.utils.chunking.**apply_per_chunk**(*func*)
Apply a function to each chunk if the input is chunked.

fletcher.algorithms.utils.chunking.**dispatch_chunked_binary_map**(*a: Any*, *b: Any*, *ops: Dict[str, Callable]*)
fletcher.algorithms.utils.chunking.**dispatch_chunked_binary_map**(*a: pyarrow.lib.ChunkedArray*, *b: Any*, *ops: Dict[str, Callable]*)
fletcher.algorithms.utils.chunking.**dispatch_chunked_binary_map**(*a: pyarrow.lib.Array*, *b: Any*, *ops: Dict[str, Callable]*)
Apply a map-like binary function where at least one of the arguments is an Arrow structure.

This will yield a pyarrow.Arrow or pyarrow.ChunkedArray as an output.

Parameters

a: scalar or np.ndarray or pa.Array or pa.ChunkedArray

b: scalar or np.ndarray or pa.Array or pa.ChunkedArray

op: dict Dictionary with the keys ('array_array', 'array_nparray', 'nparray_array', 'array_scalar', 'scalar_array')

fletcher.algorithms.utils.kmp module

Utility functions for the Knuth Moris Pratt string matching algorithm.

fletcher.algorithms.utils.kmp.**append_to_kmp_matching**(*matched_len: int*, *character: int*, *pat: bytes*, *failure_function: numpy.ndarray*) → int

Append a character to a Knuth Moris Pratt matching. This function can be used to search for *pat* in a text with the KMP algorithm. Parameters ———- matched_len: int

The length of the previous maximum prefix of *pat* that was a suffix of the text. Must sattisfy *0 <= matched_len < len(pat)*.

character: int The next character of the text.

pat: bytes The pattern that is searched in the text.

failure_function: np.ndarray The KMP failure function of *pat*. Should be obtained through *compute_kmp_failure_function(pat)*.

int The length of the maximum prefix of *pat* that is a suffix of the text after appendng *character*. Always => *0* and <= *min(matched_len + 1, len(pat))*.

fletcher.algorithms.utils.kmp.**compute_kmp_failure_function**(*pat: bytes*) → numpy.ndarray

Compute the Knuth Moris Pratt failure function. Parameters ———- pat : bytes

The bytes representation of the string for which to compute the KMP failure function.

Numpy array f of len(pat) + 1 integers. f[0] = -1. For i > 0, f[i] is equal to the length of the longest propper suffix of pat[:i] that is a prefix of pat. Since, only propper suffixes are considered, for i > 0 we have 0 <= f[i] < i.

```
>>> comp
```

Module contents

Submodules

fletcher.algorithms.bool module

fletcher.algorithms.bool.**all_op**(*arr: Union[pyarrow.lib.ChunkedArray, pyarrow.lib.Array]*, *skipna: bool*) → bool

Perform all() on a boolean Arrow structure.

fletcher.algorithms.bool.**all_true**(*arr: pyarrow.lib.Array*) → pyarrow.lib.Array

Return a boolean array with all-True, all-valid with the same size .

fletcher.algorithms.bool.**all_true_like**(*arr: pyarrow.lib.Array*) → pyarrow.lib.Array

Return a boolean array with all-True with the same size as the input and the same valid bitmap.

fletcher.algorithms.bool.**any_op**(*arr: Union[pyarrow.lib.ChunkedArray, pyarrow.lib.Array]*, *skipna: bool*) → bool

Perform any() on a boolean Arrow structure.

fletcher.algorithms.bool.**bitmap_or_unaligned**(*length: int*, *a: bytes*, *offset_a: int*, *b: bytes*, *offset_b: int*, *result: numpy.ndarray*) → None

Perform OR on two bitmaps without any alignments requirements.

fletcher.algorithms.bool.**bitmap_or_unaligned_with_numpy**(*length: int*, *valid_bits_a: bytes*, *a: bytes*, *offset_a: int*, *b: numpy.ndarray*, *result: numpy.ndarray*, *valid_bits: numpy.ndarray*) → int

Perform OR on a bitmap with valid bitmask and a numpy array with truthy rows.

fletcher.algorithms.bool.**bitmap_or_unaligned_with_numpy_nonnull**(*length: int*, *a: bytes*, *offset_a: int*, *b: numpy.ndarray*, *result: numpy.ndarray*) → None

Perform OR on a bitmap and a numpy array with truthy rows.

fletcher.algorithms.bool.**masked_bitmap_or_unaligned**(*length: int*, *valid_bits_a: bytes*, *a: bytes*, *offset_a: int*, *valid_bits_b: bytes*, *b: bytes*, *offset_b: int*, *result: numpy.ndarray*, *valid_bits: numpy.ndarray*) → int

Perform OR on two bitmaps with valid bitmasks without any alignment requirements.

fletcher.algorithms.bool.**or_array_array**(*a: pyarrow.lib.Array*, *b: pyarrow.lib.Array*) → pyarrow.lib.Array

Perform `pyarrow.Array | pyarrow.Array`.

fletcher.algorithms.bool.**or_array_nparray**(*a: pyarrow.lib.Array*, *b: numpy.ndarray*) → pyarrow.lib.Array

Perform `pa.Array | np.ndarray`.

fletcher.algorithms.bool.**or_na**(*arr: pyarrow.lib.Array*) → pyarrow.lib.Array

Apply `array | NA` with a boolean pyarrow.Array.

fletcher.algorithms.bool.**or_vectorised**(*a: Union[pyarrow.lib.Array, pyarrow.lib.ChunkedArray]*, *b: Any*)

Perform OR on a boolean Arrow structure and a second operator.

fletcher.algorithms.numpy_ufunc module

fletcher.algorithms.string module

fletcher.algorithms.string.**apply_binary_str**(*a: Union[pyarrow.lib.Array, pyarrow.lib.ChunkedArray]*, *b: Union[pyarrow.lib.Array, pyarrow.lib.ChunkedArray]*, ***, *func: Callable*, *output_dtype*, *parallel: bool = False*)

Apply an element-wise numba-jitted function on two Arrow columns.

The supplied function must return a numpy-compatible scalar. Handling of missing data and chunking of the inputs is done automatically.

fletcher.algorithms.string.**get_utf8_size**(*first_byte: int*)

fletcher.algorithms.string.**shift_unaligned_bitmap**(*valid_buffer: pyarrow.lib.Buffer*, *offset: int*, *length: int*) → pyarrow.lib.Buffer

Shift an unaligned bitmap to be offsetted at 0.

fletcher.algorithms.string_builder module

class fletcher.algorithms.string_builder.**LibcMalloc**

Bases: numba.core.types.function_type.WrapperAddressProtocol

Methods

signature()	Return the signature of a first-class function.

signature()

Return the signature of a first-class function.

Returns

sig [Signature] The returned Signature instance represents the type of a first-class function that the given WrapperAddressProtocol instance represents.

fletcher.algorithms.string_builder.**byte_for_bits**(*num_bits*)

fletcher.algorithms.string_builder.**finalize_string_array**(*sba*, *typ*) → pyarrow.lib.Array

Take a StringArrayBuilder and returns a pyarrow.StringArray. The native memory in the StringArrayBuilder is free'd during this method call and this is unusable afterwards but also doesn't leak any memory.

fletcher.algorithms.string_builder.**malloc_nojit**(*size: int*)

fletcher.algorithms.string_builder_nojit module

class fletcher.algorithms.string_builder_nojit.**BitVector**(*initial_size: int*)
Bases: object

Builder that constructs a buffer based on bit-packed chunks.

As the memory is owned by this object but we cannot override __del__, you need to explicitly call delete() to free the native memory.

Methods

expand()	Double the size of the underlying buffer and copy over the existing data.

append_false	
append_true	
delete	
get	

append_false()

append_true()

delete()

expand()
Double the size of the underlying buffer and copy over the existing data.

This allocates a new buffer and copies the data.

get(*idx*)

class fletcher.algorithms.string_builder_nojit.**ByteVector**(*initial_size: int*)
Bases: object

Builder that constructs a buffer based on byte-sized chunks.

As the memory is owned by this object but we cannot override __del__, you need to explicitly call delete() to free the native memory.

Methods

append(byte)	Append a single byte to the stream.
append_bytes(ptr, length)	Append a range of bytes.
append_int16(i16)	Append a signed 16bit integer.
append_int32(i32)	Append a signed 32bit integer.
append_int64(i64)	Append a signed 64bit integer.
append_uint32(i32)	Append an unsigned 32bit integer.
expand()	Double the size of the underlying buffer and copy over the existing data.

delete	
get_int16	
get_int32	
get_int64	
get_uint32	
get_uint8	

append(*byte*)
Append a single byte to the stream.

append_bytes(*ptr*, *length*)
Append a range of bytes.

append_int16(*i16*)
Append a signed 16bit integer.

append_int32(*i32*)
Append a signed 32bit integer.

append_int64(*i64*)
Append a signed 64bit integer.

append_uint32(*i32*)
Append an unsigned 32bit integer.

delete()

expand()
Double the size of the underlying buffer and copy over the existing data.

This allocates a new buffer and copies the data.

get_int16(*idx*)

get_int32(*idx*)

get_int64(*idx*)

get_uint32(*idx*)

get_uint8(*idx*)

class fletcher.algorithms.string_builder_nojit.**StringArrayBuilder**(*initial_size: int*)

Bases: object

Numba-based builder to construct pyarrow.StringArray instances.

As Numba doesn't allow us to override __del__, we must always call delete to free up the used (native) memory.

Methods

append_null	
append_value	
delete	

append_null()

append_value(*ptr*, *length*)

delete()

fletcher.algorithms.string_builder_nojit.**byte_for_bits**(*num_bits*)

fletcher.algorithms.string_builder_nojit.**finalize_string_array**(*sba*, *typ*) → pyarrow.lib.Array

Take a StringArrayBuilder and returns a pyarrow.StringArray. The native memory in the StringArrayBuilder is free'd during this method call and this is unusable afterwards but also doesn't leak any memory.

Module contents

3.1.2 Submodules

fletcher.base module

class fletcher.base.**FletcherBaseArray**

Bases: *fletcher.string_mixin.StringSupportingExtensionArray*

Pandas ExtensionArray implementation base backed by an Apache Arrow structure.

Attributes

T

base Return base object of the underlying data.

dtype Return the ExtensionDtype of this array.

nbytes The number of bytes needed to store this object in memory.

ndim Return the number of dimensions of the underlying data.

shape Return the shape of the data.

size Return the number of elements in this array.

Methods

all([skipna])	Compute whether all boolean values are True.
any([skipna])	Compute whether any boolean value is True.
argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind, na_position])	Return the indices that would sort this array.
astype(dtype[, copy])	Cast to a NumPy array with 'dtype'.
copy()	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
isna()	Boolean NumPy array indicating if each value is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.

continues on next page

Table 4 – continued from previous page

shift([periods, fill_value])	Shift values by desired number.
sum([skipna])	Return the sum of the values.
take(indices, *[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
transpose(*axes)	Return a transposed view on this array.
unique()	Compute the ExtensionArray of unique values.
value_counts([dropna])	Return a Series containing counts of each unique value.
view([dtype])	Return a view on the array.

all(*skipna: bool = False*) → Optional[bool]
Compute whether all boolean values are True.

any(*skipna: bool = False*, ***kwargs*) → Optional[bool]
Compute whether any boolean value is True.

astype(*dtype*, *copy=True*)
Cast to a NumPy array with 'dtype'.

Parameters

dtype [str or dtype] Typecode or data-type to which the array is cast.

copy [bool, default True] Whether to copy the data, even if not necessary. If False, a copy is made only if the old dtype does not match the new dtype.

Returns

array [ndarray] NumPy ndarray with 'dtype' for its dtype.

property base
Return base object of the underlying data.

property dtype
Return the ExtensionDtype of this array.

isna() → numpy.ndarray
Boolean NumPy array indicating if each value is missing.

This should return a 1-D array the same length as 'self'.

property ndim
Return the number of dimensions of the underlying data.

property shape
Return the shape of the data.

property size
Return the number of elements in this array.

Returns

size [int]

sum(*skipna: bool = True*)
Return the sum of the values.

unique()
Compute the ExtensionArray of unique values.

It relies on the Pyarrow.ChunkedArray.unique and if it fails, comes back to the naive implementation.

Returns

uniques [ExtensionArray]

value_counts(*dropna: bool = True*) → pandas.core.series.Series
Return a Series containing counts of each unique value.

Parameters

dropna [bool, default True] Don't include counts of missing values.

Returns

counts [Series]

See also:

Series.value_counts

class fletcher.base.**FletcherBaseDtype**(*arrow_dtype: pyarrow.lib.DataType*)
Bases: pandas.core.dtypes.base.ExtensionDtype

Dtype base for a pandas ExtensionArray backed by an Apache Arrow structure.

Attributes

itemsize

kind Return a character code (one of 'biufcmMOSUV'), default 'O'.

name Return a string identifying the data type.

names Ordered list of field names, or None if there are no fields.

type Return the scalar type for the array, e.g.

Methods

construct_array_type()	Return the array type associated with this dtype.
construct_from_string(string)	Construct this type from a string.
example()	Get a simple array with example content.
is_dtype(dtype)	Check if we match 'dtype'.

example()
Get a simple array with example content.

property itemsize

property kind
Return a character code (one of 'biufcmMOSUV'), default 'O'.

This should match the NumPy dtype used when the array is converted to an ndarray, which is probably 'O' for object if the extension type cannot be represented as a built-in NumPy type.

See also:

numpy.dtype.kind

na_value = <NA>

property name
Return a string identifying the data type.

Will be used for display in, e.g. `Series.dtype`

property type
Return the scalar type for the array, e.g. `int`.

It's expected `ExtensionArray[item]` returns an instance of `ExtensionDtype.type` for scalar `item`.

class fletcher.base.**FletcherChunkedArray**(*array*, *dtype=None*, *copy=None*)
Bases: *fletcher.base.FletcherBaseArray*

Pandas ExtensionArray implementation backed by Apache Arrow.

Attributes

T

base Return base object of the underlying data.

dtype Return the ExtensionDtype of this array.

nbytes Return the number of bytes needed to store this object in memory.

ndim Return the number of dimensions of the underlying data.

shape Return the shape of the data.

size Return the number of elements in this array.

Methods

all([skipna])	Compute whether all boolean values are True.
any([skipna])	Compute whether any boolean value is True.
argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind, na_position])	Return the indices that would sort this array.
astype(dtype[, copy])	Cast to a NumPy array with 'dtype'.
copy()	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
flatten()	Flatten the array.
isna()	Boolean NumPy array indicating if each value is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
shift([periods, fill_value])	Shift values by desired number.
sum([skipna])	Return the sum of the values.
take(indices[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
transpose(*axes)	Return a transposed view on this array.

continues on next page

Table 6 – continued from previous page

`unique`()	Compute the ExtensionArray of unique values.
`value_counts`([dropna])	Return a Series containing counts of each unique value.
`view`([dtype])	Return a view on the array.

copy() → pandas.core.arrays.base.ExtensionArray
Return a copy of the array.

Parameters

deep [bool, default False] Also copy the underlying data backing this array.

Returns

ExtensionArray

factorize(*na_sentinel=- 1*)
Encode the extension array as an enumerated type.

Parameters

na_sentinel [int, default -1] Value to use in the *codes* array to indicate missing values.

Returns

codes [ndarray] An integer NumPy array that's an indexer into the original ExtensionArray.

uniques [ExtensionArray] An ExtensionArray containing the unique values of *self*.

Note: uniques will *not* contain an entry for the NA value of the ExtensionArray if there are any missing values present in *self*.

See also:

factorize Top-level factorize method that dispatches here.

Notes

`pandas.factorize()` offers a *sort* keyword as well.

fillna(*value=None*, *method=None*, *limit=None*)
Fill NA/NaN values using the specified method.

Parameters

value [scalar, array-like] If a scalar value is passed it is used to fill all missing values. Alternatively, an array-like 'value' can be given. It's expected that the array-like have the same length as 'self'.

method [{'backfill', 'bfill', 'pad', 'ffill', None}, default None] Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap

limit [int, default None] If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled.

Returns

filled [ExtensionArray with NA/NaN filled]

flatten()
Flatten the array.

property nbytes
Return the number of bytes needed to store this object in memory.

take(*indices: Union[Sequence[int], numpy.ndarray]*, *allow_fill: bool = False*, *fill_value: Optional[Any] = None*) → pandas.core.arrays.base.ExtensionArray
Take elements from an array.

Parameters

indices [sequence of integers] Indices to be taken.

allow_fill [bool, default False] How to handle negative values in *indices*. * False: negative values in *indices* indicate positional indices

from the right (the default). This is similar to `numpy.take()`.

- True: negative values in *indices* indicate missing values. These values are set to *fill_value*. Any other other negative values raise a `ValueError`.

fill_value [any, optional] Fill value to use for NA-indices when *allow_fill* is True. This may be `None`, in which case the default NA value for the type, `self.dtype.na_value`, is used. For many ExtensionArrays, there will be two representations of *fill_value*: a user-facing "boxed" scalar, and a low-level physical NA value. *fill_value* should be the user-facing version, and the implementation should handle translating that to the physical version for processing the take if nescessary.

Returns

ExtensionArray

Raises

IndexError When the indices are out of bounds for the array.

ValueError When *indices* contains negative values other than `-1` and *allow_fill* is True.

See also:

numpy.take

pandas.api.extensions.take

Notes

ExtensionArray.take is called by `Series.__getitem__`, `.loc`, `iloc`, when *indices* is a sequence of values. Additionally, it's called by `Series.reindex()`, or any other method that causes realignemnt, with a *fill_value*.

class fletcher.base.**FletcherChunkedDtype**(*arrow_dtype: pyarrow.lib.DataType*)
Bases: *fletcher.base.FletcherBaseDtype*

Dtype for a pandas ExtensionArray backed by Apache Arrow's pyarrow.ChunkedArray.

Attributes

itemsize

kind Return a character code (one of 'biufcmMOSUV'), default 'O'.

name Return a string identifying the data type.

names Ordered list of field names, or None if there are no fields.

type Return the scalar type for the array, e.g.

Methods

construct_array_type(*args)	Return the array type associated with this dtype.
construct_from_string(string)	Attempt to construct this type from a string.
example()	Get a simple array with example content.
is_dtype(dtype)	Check if we match 'dtype'.

classmethod construct_array_type(**args*) → Type[*fletcher.base.FletcherChunkedArray*]
Return the array type associated with this dtype.

Returns

type

classmethod construct_from_string(*string: str*) → *fletcher.base.FletcherChunkedDtype*
Attempt to construct this type from a string.

Parameters

string [str]

Returns

self [instance of 'cls']

Raises

TypeError If a class cannot be constructed from this 'string'.

Examples

If the extension dtype can be constructed without any arguments, the following may be an adequate implementation. >>> @classmethod ... def construct_from_string(cls, string) ... if string == cls.name: ... return cls() ... else: ... raise TypeError("Cannot construct a '{}' from " ... "'{}'".format(cls, string))

class fletcher.base.**FletcherContinuousArray**(*array*, *dtype=None*, *copy: Optional[bool] = None*)
Bases: *fletcher.base.FletcherBaseArray*

Pandas ExtensionArray implementation backed by Apache Arrow's pyarrow.Array.

Attributes

T

base Return base object of the underlying data.

dtype Return the ExtensionDtype of this array.

nbytes Return the number of bytes needed to store this object in memory.

ndim Return the number of dimensions of the underlying data.

shape Return the shape of the data.

size Return the number of elements in this array.

Methods

all([skipna])	Compute whether all boolean values are True.
any([skipna])	Compute whether any boolean value is True.
argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind, na_position])	Return the indices that would sort this array.
astype(dtype[, copy])	Cast to a NumPy array with 'dtype'.
copy()	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
flatten()	Flatten the array.
isna()	Boolean NumPy array indicating if each value is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
shift([periods, fill_value])	Shift values by desired number.
sum([skipna])	Return the sum of the values.
take(indices[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
transpose(*axes)	Return a transposed view on this array.
unique()	Compute the ExtensionArray of unique values.
value_counts([dropna])	Return a Series containing counts of each unique value.
view([dtype])	Return a view on the array.

copy() → pandas.core.arrays.base.ExtensionArray

Return a copy of the array.

Currently is a shadow copy - pyarrow array are supposed to be immutable.

Returns

ExtensionArray

factorize(*na_sentinel=- 1*)

Encode the extension array as an enumerated type.

Parameters

na_sentinel [int, default -1] Value to use in the *codes* array to indicate missing values.

Returns

codes [ndarray] An integer NumPy array that's an indexer into the original ExtensionArray.

uniques [ExtensionArray] An ExtensionArray containing the unique values of *self*.

Note: uniques will *not* contain an entry for the NA value of the ExtensionArray if there are any missing values present in *self*.

See also:

factorize Top-level factorize method that dispatches here.

Notes

`pandas.factorize()` offers a *sort* keyword as well.

fillna(*value=None*, *method=None*, *limit=None*)
Fill NA/NaN values using the specified method.

Parameters

value [scalar, array-like] If a scalar value is passed it is used to fill all missing values. Alternatively, an array-like 'value' can be given. It's expected that the array-like have the same length as 'self'.

method [{'backfill', 'bfill', 'pad', 'ffill', None}, default None] Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap

limit [int, default None] If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled.

Returns

filled [ExtensionArray with NA/NaN filled]

flatten()
Flatten the array.

property nbytes
Return the number of bytes needed to store this object in memory.

take(*indices: Union[Sequence[int], numpy.ndarray]*, *allow_fill: bool = False*, *fill_value: Optional[Any] = None*) → pandas.core.arrays.base.ExtensionArray
Take elements from an array.

Parameters

indices [sequence of integers] Indices to be taken.

allow_fill [bool, default False] How to handle negative values in *indices*. * False: negative values in *indices* indicate positional indices

from the right (the default). This is similar to `numpy.take()`.

- True: negative values in *indices* indicate missing values. These values are set to *fill_value*. Any other other negative values raise a `ValueError`.

fill_value [any, optional] Fill value to use for NA-indices when *allow_fill* is True. This may be `None`, in which case the default NA value for the type, `self.dtype.na_value`, is used. For many ExtensionArrays, there will be two representations of *fill_value*: a user-facing "boxed" scalar, and a low-level physical NA value. *fill_value* should be the user-facing version, and the implementation should handle translating that to the physical version for processing the take if nescessary.

Returns

ExtensionArray

Raises

IndexError When the indices are out of bounds for the array.

ValueError When *indices* contains negative values other than `-1` and *allow_fill* is True.

See also:

numpy.take

pandas.api.extensions.take

Notes

ExtensionArray.take is called by `Series.__getitem__`, `.loc`, `iloc`, when *indices* is a sequence of values. Additionally, it's called by `Series.reindex()`, or any other method that causes realignemnt, with a *fill_value*.

class fletcher.base.**FletcherContinuousDtype**(*arrow_dtype: pyarrow.lib.DataType*)

Bases: *fletcher.base.FletcherBaseDtype*

Dtype for a pandas ExtensionArray backed by Apache Arrow's pyarrow.Array.

Attributes

itemsize

kind Return a character code (one of 'biufcmMOSUV'), default 'O'.

name Return a string identifying the data type.

names Ordered list of field names, or None if there are no fields.

type Return the scalar type for the array, e.g.

Methods

construct_array_type(*args)	Return the array type associated with this dtype.
construct_from_string(string)	Attempt to construct this type from a string.
example()	Get a simple array with example content.
is_dtype(dtype)	Check if we match 'dtype'.

classmethod construct_array_type(**args*)

Return the array type associated with this dtype.

Returns

type

classmethod construct_from_string(*string: str*)

Attempt to construct this type from a string.

Parameters

string

Returns

self [instance of 'cls']

Raises

TypeError If a class cannot be constructed from this 'string'.

Examples

If the extension dtype can be constructed without any arguments, the following may be an adequate implementation. >>> @classmethod ... def construct_from_string(cls, string) ... if string == cls.name: ... return cls() ... else: ... raise TypeError("Cannot construct a '{}' from " ... "'{}'".format(cls, string))

`fletcher.base.`**`pandas_from_arrow`**(*arrow_object: Union[pyarrow.lib.RecordBatch, pyarrow.lib.Table, pyarrow.lib.Array, pyarrow.lib.ChunkedArray]*, *continuous: bool = False*)

Convert Arrow object instance to their Pandas equivalent by using Fletcher.

The conversion rules are:

- {RecordBatch, Table} -> DataFrame
- {Array, ChunkedArray} -> Series

Parameters

arrow_object [RecordBatch, Table, Array or ChunkedArray] object to be converted

continuous [bool] Use FletcherContinuousArray instead of FletcherChunkedArray

fletcher.io module

`fletcher.io.`**`read_parquet`**(*path*, *columns: Optional[List[str]] = None*, *continuous: bool = False*) → pandas.core.frame.DataFrame

Load a parquet object from the file path, returning a DataFrame with fletcher columns.

Parameters

path [str or file-like]

continuous [bool] Use FletcherContinuousArray instead of FletcherChunkedArray

Returns

pd.DataFrame

fletcher.string_array module

class `fletcher.string_array.`**`TextAccessor`**(*obj*)

Bases: *fletcher.string_array.TextAccessorBase*

Accessor for pandas exposed as `.fr_strx`.

Methods

cat(others)	Concatenate strings in the Series/Index with given separator.
contains(pat[, case, regex])	Test if pattern or regex is contained within a string of a Series or Index.
endswith(pat)	Check whether a row ends with a certain pattern.
replace(pat, repl[, n, case, regex])	Replace occurrences of pattern/regex in the Series/Index with some other string.

continues on next page

Table 10 – continued from previous page

slice([start, end, step])	Extract every *step* character from strings from *start* to *end*.
startswith(pat)	Check whether a row starts with a certain pattern.
strip([to_strip])	Strip whitespaces from both ends of strings.
zfill(width)	Pad strings in the Series/Index by prepending '0' characters.

count	
isalnum	
isalpha	
isdecimal	
isdigit	
islower	
isnumeric	
isspace	
istitle	
isupper	

cat (*others: Optional[*fletcher.base.FletcherBaseArray*]*) → pandas.core.series.Series
Concatenate strings in the Series/Index with given separator.

If *others* is specified, this function concatenates the Series/Index and elements of *others* element-wise. If *others* is not passed, then all values in the Series/Index are concatenated into a single string with a given *sep*.

contains (*pat: str*, *case: bool = True*, *regex: bool = True*) → pandas.core.series.Series
Test if pattern or regex is contained within a string of a Series or Index.

Return boolean Series or Index based on whether a given pattern or regex is contained within a string of a Series or Index.

This implementation differs to the one in `pandas`:

- We always return a missing for missing data.
- You cannot pass flags for the regular expression module.

Parameters

pat [str] Character sequence or regular expression.

case [bool, default True] If True, case sensitive.

regex [bool, default True] If True, assumes the pat is a regular expression.

If False, treats the pat as a literal string.

Returns

Series or Index of boolean values A Series or Index of boolean values indicating whether the given pattern is contained within the string of each element of the Series or Index.

count (*pat: str*, *regex: bool = True*) → pandas.core.series.Series

endswith (*pat*)
Check whether a row ends with a certain pattern.

isalnum ()

isalpha()

isdecimal()

isdigit()

islower()

isnumeric()

isspace()

istitle()

isupper()

replace(*pat: str*, *repl: str*, *n: int = - 1*, *case: bool = True*, *regex: bool = True*)
Replace occurrences of pattern/regex in the Series/Index with some other string. Equivalent to str.replace() or re.sub().

Return string Series where in each row the occurrences of the given pattern or regex `pat` are replaced by `repl`.

This implementation differs to the one in `pandas`:

- We always return a missing for missing data.
- You cannot pass flags for the regular expression module.

Parameters

pat [str] Character sequence or regular expression.

repl [str] Replacement string.

n [int] Number of replacements to make from start.

case [bool, default True] If True, case sensitive.

regex [bool, default True] If True, assumes the pat is a regular expression. If False, treats the pat as a literal string.

Returns

Series of string values.

slice(*start=0*, *end=None*, *step=1*)
Extract every *step* character from strings from *start* to *end*.

startswith(*pat*)
Check whether a row starts with a certain pattern.

strip(*to_strip=None*)
Strip whitespaces from both ends of strings.

zfill(*width: int*) → pandas.core.series.Series
Pad strings in the Series/Index by prepending '0' characters.

class fletcher.string_array.**TextAccessorBase**(*obj*)
Bases: object

Base class for `.fr_str` and `.fr_strx` accessors.

class fletcher.string_array.**TextAccessorExt**(*obj*)
Bases: *fletcher.string_array.TextAccessorBase*

Accessor for pandas exposed as `.fr_str`.

fletcher.string_array.**buffers_as_arrays**(*sa*)

fletcher.string_mixin module

class fletcher.string_mixin.**StringSupportingExtensionArray**
Bases: fletcher.string_mixin._IntermediateExtensionArray

Attributes

T

dtype An instance of 'ExtensionDtype'.

nbytes The number of bytes needed to store this object in memory.

ndim Extension Arrays are only allowed to be 1-dimensional.

shape Return a tuple of the array dimensions.

size The number of elements in the array.

Methods

argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind, na_position])	Return the indices that would sort this array.
astype(dtype[, copy])	Cast to a NumPy array with 'dtype'.
copy()	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
isna()	A 1-D array indicating if each value is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
shift([periods, fill_value])	Shift values by desired number.
take(indices, *[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
transpose(*axes)	Return a transposed view on this array.
unique()	Compute the ExtensionArray of unique values.
view([dtype])	Return a view on the array.

fletcher.testing module

fletcher.testing.**examples**(*example_list: List[Any]*, *example_kword: str*)
Annotation for tests using hypothesis input generation.

It is similar to the @example annotation but allows specifying a list of examples.

Parameters

example_list: list of examples

example_kword: which parameter to use for passing the example values to the test function

Returns

method: wrapper method

3.1.3 Module contents

class fletcher.**FletcherBaseArray**
Bases: *fletcher.string_mixin.StringSupportingExtensionArray*

Pandas ExtensionArray implementation base backed by an Apache Arrow structure.

Attributes

T

base Return base object of the underlying data.

dtype Return the ExtensionDtype of this array.

nbytes The number of bytes needed to store this object in memory.

ndim Return the number of dimensions of the underlying data.

shape Return the shape of the data.

size Return the number of elements in this array.

Methods

all([skipna])	Compute whether all boolean values are True.
any([skipna])	Compute whether any boolean value is True.
argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind, na_position])	Return the indices that would sort this array.
astype(dtype[, copy])	Cast to a NumPy array with 'dtype'.
copy()	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
isna()	Boolean NumPy array indicating if each value is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.

continues on next page

Table 12 – continued from previous page

searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
shift([periods, fill_value])	Shift values by desired number.
sum([skipna])	Return the sum of the values.
take(indices, *[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
transpose(*axes)	Return a transposed view on this array.
unique()	Compute the ExtensionArray of unique values.
value_counts([dropna])	Return a Series containing counts of each unique value.
view([dtype])	Return a view on the array.

all(*skipna: bool = False*) → Optional[bool]
Compute whether all boolean values are True.

any(*skipna: bool = False*, ***kwargs*) → Optional[bool]
Compute whether any boolean value is True.

astype(*dtype*, *copy=True*)
Cast to a NumPy array with 'dtype'.

Parameters

dtype [str or dtype] Typecode or data-type to which the array is cast.

copy [bool, default True] Whether to copy the data, even if not necessary. If False, a copy is made only if the old dtype does not match the new dtype.

Returns

array [ndarray] NumPy ndarray with 'dtype' for its dtype.

property base
Return base object of the underlying data.

property dtype
Return the ExtensionDtype of this array.

isna() → numpy.ndarray
Boolean NumPy array indicating if each value is missing.

This should return a 1-D array the same length as 'self'.

property ndim
Return the number of dimensions of the underlying data.

property shape
Return the shape of the data.

property size
Return the number of elements in this array.

Returns

size [int]

sum(*skipna: bool = True*)
Return the sum of the values.

unique()
Compute the ExtensionArray of unique values.

It relies on the Pyarrow.ChunkedArray.unique and if it fails, comes back to the naive implementation.

Returns

uniques [ExtensionArray]

value_counts(*dropna: bool = True*) → pandas.core.series.Series
Return a Series containing counts of each unique value.

Parameters

dropna [bool, default True] Don't include counts of missing values.

Returns

counts [Series]

See also:

Series.value_counts

class fletcher.**FletcherBaseDtype**(*arrow_dtype: pyarrow.lib.DataType*)
Bases: pandas.core.dtypes.base.ExtensionDtype

Dtype base for a pandas ExtensionArray backed by an Apache Arrow structure.

Attributes

itemsize

kind Return a character code (one of 'biufcmMOSUV'), default 'O'.

name Return a string identifying the data type.

names Ordered list of field names, or None if there are no fields.

type Return the scalar type for the array, e.g.

Methods

construct_array_type()	Return the array type associated with this dtype.
construct_from_string(string)	Construct this type from a string.
example()	Get a simple array with example content.
is_dtype(dtype)	Check if we match 'dtype'.

example()
Get a simple array with example content.

property itemsize

property kind
Return a character code (one of 'biufcmMOSUV'), default 'O'.

This should match the NumPy dtype used when the array is converted to an ndarray, which is probably 'O' for object if the extension type cannot be represented as a built-in NumPy type.

See also:

numpy.dtype.kind

na_value = <NA>

property name
Return a string identifying the data type.

Will be used for display in, e.g. `Series.dtype`

property type
Return the scalar type for the array, e.g. `int`.

It's expected `ExtensionArray[item]` returns an instance of `ExtensionDtype.type` for scalar `item`.

class fletcher.**FletcherChunkedArray**(*array*, *dtype=None*, *copy=None*)
Bases: *fletcher.base.FletcherBaseArray*

Pandas ExtensionArray implementation backed by Apache Arrow.

Attributes

T

base Return base object of the underlying data.

dtype Return the ExtensionDtype of this array.

nbytes Return the number of bytes needed to store this object in memory.

ndim Return the number of dimensions of the underlying data.

shape Return the shape of the data.

size Return the number of elements in this array.

Methods

all([skipna])	Compute whether all boolean values are True.
any([skipna])	Compute whether any boolean value is True.
argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind, na_position])	Return the indices that would sort this array.
astype(dtype[, copy])	Cast to a NumPy array with 'dtype'.
copy()	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
flatten()	Flatten the array.
isna()	Boolean NumPy array indicating if each value is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
shift([periods, fill_value])	Shift values by desired number.
sum([skipna])	Return the sum of the values.
take(indices[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
transpose(*axes)	Return a transposed view on this array.

continues on next page

Table 14 – continued from previous page

unique()	Compute the ExtensionArray of unique values.
value_counts([dropna])	Return a Series containing counts of each unique value.
view([dtype])	Return a view on the array.

copy() → pandas.core.arrays.base.ExtensionArray
Return a copy of the array.

Parameters

deep [bool, default False] Also copy the underlying data backing this array.

Returns

ExtensionArray

factorize(*na_sentinel=- 1*)
Encode the extension array as an enumerated type.

Parameters

na_sentinel [int, default -1] Value to use in the *codes* array to indicate missing values.

Returns

codes [ndarray] An integer NumPy array that's an indexer into the original ExtensionArray.

uniques [ExtensionArray] An ExtensionArray containing the unique values of *self*.

Note: uniques will *not* contain an entry for the NA value of the ExtensionArray if there are any missing values present in *self*.

See also:

factorize Top-level factorize method that dispatches here.

Notes

pandas.factorize() offers a *sort* keyword as well.

fillna(*value=None*, *method=None*, *limit=None*)
Fill NA/NaN values using the specified method.

Parameters

value [scalar, array-like] If a scalar value is passed it is used to fill all missing values. Alternatively, an array-like 'value' can be given. It's expected that the array-like have the same length as 'self'.

method [{'backfill', 'bfill', 'pad', 'ffill', None}, default None] Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap

limit [int, default None] If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled.

Returns

filled [ExtensionArray with NA/NaN filled]

flatten()
Flatten the array.

property nbytes
Return the number of bytes needed to store this object in memory.

take(*indices: Union[Sequence[int], numpy.ndarray]*, *allow_fill: bool = False*, *fill_value: Optional[Any] = None*) → pandas.core.arrays.base.ExtensionArray
Take elements from an array.

Parameters

indices [sequence of integers] Indices to be taken.

allow_fill [bool, default False] How to handle negative values in *indices*. * False: negative values in *indices* indicate positional indices

from the right (the default). This is similar to `numpy.take()`.

- True: negative values in *indices* indicate missing values. These values are set to *fill_value*. Any other other negative values raise a `ValueError`.

fill_value [any, optional] Fill value to use for NA-indices when *allow_fill* is True. This may be `None`, in which case the default NA value for the type, `self.dtype.na_value`, is used. For many ExtensionArrays, there will be two representations of *fill_value*: a user-facing "boxed" scalar, and a low-level physical NA value. *fill_value* should be the user-facing version, and the implementation should handle translating that to the physical version for processing the take if nescessary.

Returns

ExtensionArray

Raises

IndexError When the indices are out of bounds for the array.

ValueError When *indices* contains negative values other than `-1` and *allow_fill* is True.

See also:

numpy.take

pandas.api.extensions.take

Notes

ExtensionArray.take is called by `Series.__getitem__`, `.loc`, `iloc`, when *indices* is a sequence of values. Additionally, it's called by `Series.reindex()`, or any other method that causes realignemnt, with a *fill_value*.

class fletcher.**FletcherChunkedDtype**(*arrow_dtype: pyarrow.lib.DataType*)
Bases: *fletcher.base.FletcherBaseDtype*

Dtype for a pandas ExtensionArray backed by Apache Arrow's pyarrow.ChunkedArray.

Attributes

itemsize

kind Return a character code (one of 'biufcmMOSUV'), default 'O'.

name Return a string identifying the data type.

names Ordered list of field names, or None if there are no fields.

type Return the scalar type for the array, e.g.

Methods

construct_array_type(*args)	Return the array type associated with this dtype.
construct_from_string(string)	Attempt to construct this type from a string.
example()	Get a simple array with example content.
is_dtype(dtype)	Check if we match 'dtype'.

classmethod construct_array_type(**args*) → Type[*fletcher.base.FletcherChunkedArray*]
Return the array type associated with this dtype.

Returns

type

classmethod construct_from_string(*string: str*) → *fletcher.base.FletcherChunkedDtype*
Attempt to construct this type from a string.

Parameters

string [str]

Returns

self [instance of 'cls']

Raises

TypeError If a class cannot be constructed from this 'string'.

Examples

If the extension dtype can be constructed without any arguments, the following may be an adequate implementation. >>> @classmethod ... def construct_from_string(cls, string) ... if string == cls.name: ... return cls() ... else: ... raise TypeError("Cannot construct a '{}' from " ... "'{}'".format(cls, string))

class fletcher.**FletcherContinuousArray**(*array*, *dtype=None*, *copy: Optional[bool] = None*)
Bases: *fletcher.base.FletcherBaseArray*

Pandas ExtensionArray implementation backed by Apache Arrow's pyarrow.Array.

Attributes

T

base Return base object of the underlying data.

dtype Return the ExtensionDtype of this array.

nbytes Return the number of bytes needed to store this object in memory.

ndim Return the number of dimensions of the underlying data.

shape Return the shape of the data.

size Return the number of elements in this array.

Methods

all([skipna])	Compute whether all boolean values are True.
any([skipna])	Compute whether any boolean value is True.
argmax()	Return the index of maximum value.
argmin()	Return the index of minimum value.
argsort([ascending, kind, na_position])	Return the indices that would sort this array.
astype(dtype[, copy])	Cast to a NumPy array with 'dtype'.
copy()	Return a copy of the array.
dropna()	Return ExtensionArray without NA values.
equals(other)	Return if another array is equivalent to this array.
factorize([na_sentinel])	Encode the extension array as an enumerated type.
fillna([value, method, limit])	Fill NA/NaN values using the specified method.
flatten()	Flatten the array.
isna()	Boolean NumPy array indicating if each value is missing.
ravel([order])	Return a flattened view on this array.
repeat(repeats[, axis])	Repeat elements of a ExtensionArray.
searchsorted(value[, side, sorter])	Find indices where elements should be inserted to maintain order.
shift([periods, fill_value])	Shift values by desired number.
sum([skipna])	Return the sum of the values.
take(indices[, allow_fill, fill_value])	Take elements from an array.
to_numpy([dtype, copy, na_value])	Convert to a NumPy ndarray.
transpose(*axes)	Return a transposed view on this array.
unique()	Compute the ExtensionArray of unique values.
value_counts([dropna])	Return a Series containing counts of each unique value.
view([dtype])	Return a view on the array.

copy() → pandas.core.arrays.base.ExtensionArray

Return a copy of the array.

Currently is a shadow copy - pyarrow array are supposed to be immutable.

Returns

ExtensionArray

factorize(*na_sentinel=- 1*)

Encode the extension array as an enumerated type.

Parameters

na_sentinel [int, default -1] Value to use in the *codes* array to indicate missing values.

Returns

codes [ndarray] An integer NumPy array that's an indexer into the original ExtensionArray.

uniques [ExtensionArray] An ExtensionArray containing the unique values of *self*.

Note: uniques will *not* contain an entry for the NA value of the ExtensionArray if there are any missing values present in *self*.

See also:

factorize Top-level factorize method that dispatches here.

Notes

`pandas.factorize()` offers a *sort* keyword as well.

fillna(*value=None*, *method=None*, *limit=None*)
Fill NA/NaN values using the specified method.

Parameters

value [scalar, array-like] If a scalar value is passed it is used to fill all missing values. Alternatively, an array-like 'value' can be given. It's expected that the array-like have the same length as 'self'.

method [{'backfill', 'bfill', 'pad', 'ffill', None}, default None] Method to use for filling holes in reindexed Series pad / ffill: propagate last valid observation forward to next valid backfill / bfill: use NEXT valid observation to fill gap

limit [int, default None] If method is specified, this is the maximum number of consecutive NaN values to forward/backward fill. In other words, if there is a gap with more than this number of consecutive NaNs, it will only be partially filled. If method is not specified, this is the maximum number of entries along the entire axis where NaNs will be filled.

Returns

filled [ExtensionArray with NA/NaN filled]

flatten()
Flatten the array.

property nbytes
Return the number of bytes needed to store this object in memory.

take(*indices: Union[Sequence[int], numpy.ndarray]*, *allow_fill: bool = False*, *fill_value: Optional[Any] = None*) → pandas.core.arrays.base.ExtensionArray
Take elements from an array.

Parameters

indices [sequence of integers] Indices to be taken.

allow_fill [bool, default False] How to handle negative values in *indices*. * False: negative values in *indices* indicate positional indices

from the right (the default). This is similar to `numpy.take()`.

- True: negative values in *indices* indicate missing values. These values are set to *fill_value*. Any other other negative values raise a `ValueError`.

fill_value [any, optional] Fill value to use for NA-indices when *allow_fill* is True. This may be `None`, in which case the default NA value for the type, `self.dtype.na_value`, is used. For many ExtensionArrays, there will be two representations of *fill_value*: a user-facing "boxed" scalar, and a low-level physical NA value. *fill_value* should be the user-facing version, and the implementation should handle translating that to the physical version for processing the take if nescessary.

Returns

ExtensionArray

Raises

IndexError When the indices are out of bounds for the array.

ValueError When *indices* contains negative values other than `-1` and *allow_fill* is True.

See also:

`numpy.take`

`pandas.api.extensions.take`

Notes

ExtensionArray.take is called by `Series.__getitem__`, `.loc`, `iloc`, when *indices* is a sequence of values. Additionally, it's called by `Series.reindex()`, or any other method that causes realignemnt, with a *fill_value*.

class fletcher.**FletcherContinuousDtype**(*arrow_dtype: pyarrow.lib.DataType*)

Bases: *fletcher.base.FletcherBaseDtype*

Dtype for a pandas ExtensionArray backed by Apache Arrow's pyarrow.Array.

Attributes

itemsize

kind Return a character code (one of 'biufcmMOSUV'), default 'O'.

name Return a string identifying the data type.

names Ordered list of field names, or None if there are no fields.

type Return the scalar type for the array, e.g.

Methods

construct_array_type(*args)	Return the array type associated with this dtype.
construct_from_string(string)	Attempt to construct this type from a string.
example()	Get a simple array with example content.
is_dtype(dtype)	Check if we match 'dtype'.

classmethod construct_array_type(**args*)

Return the array type associated with this dtype.

Returns

type

classmethod construct_from_string(*string: str*)

Attempt to construct this type from a string.

Parameters

string

Returns

self [instance of 'cls']

Raises

TypeError If a class cannot be constructed from this 'string'.

Examples

If the extension dtype can be constructed without any arguments, the following may be an adequate implementation. >>> @classmethod ... def construct_from_string(cls, string) ... if string == cls.name: ... return cls() ... else: ... raise TypeError("Cannot construct a '{}' from " ... "'{}'".format(cls, string))

class fletcher.**TextAccessor**(*obj*)

Bases: *fletcher.string_array.TextAccessorBase*

Accessor for pandas exposed as `.fr_strx`.

Methods

cat(others)	Concatenate strings in the Series/Index with given separator.
contains(pat[, case, regex])	Test if pattern or regex is contained within a string of a Series or Index.
endswith(pat)	Check whether a row ends with a certain pattern.
replace(pat, repl[, n, case, regex])	Replace occurrences of pattern/regex in the Series/Index with some other string.
slice([start, end, step])	Extract every *step* character from strings from *start* to *end*.
startswith(pat)	Check whether a row starts with a certain pattern.
strip([to_strip])	Strip whitespaces from both ends of strings.
zfill(width)	Pad strings in the Series/Index by prepending '0' characters.

count	
isalnum	
isalpha	
isdecimal	
isdigit	
islower	
isnumeric	
isspace	
istitle	
isupper	

cat(*others: Optional[fletcher.base.FletcherBaseArray]*) → pandas.core.series.Series

Concatenate strings in the Series/Index with given separator.

If *others* is specified, this function concatenates the Series/Index and elements of *others* element-wise. If *others* is not passed, then all values in the Series/Index are concatenated into a single string with a given *sep*.

contains(*pat: str*, *case: bool = True*, *regex: bool = True*) → pandas.core.series.Series

Test if pattern or regex is contained within a string of a Series or Index.

Return boolean Series or Index based on whether a given pattern or regex is contained within a string of a Series or Index.

This implementation differs to the one in `pandas`:

- We always return a missing for missing data.

- You cannot pass flags for the regular expression module.

Parameters

pat [str] Character sequence or regular expression.

case [bool, default True] If True, case sensitive.

regex [bool, default True] If True, assumes the pat is a regular expression.

If False, treats the pat as a literal string.

Returns

Series or Index of boolean values A Series or Index of boolean values indicating whether the given pattern is contained within the string of each element of the Series or Index.

count(*pat: str*, *regex: bool = True*) → pandas.core.series.Series

endswith(*pat*)

Check whether a row ends with a certain pattern.

isalnum()

isalpha()

isdecimal()

isdigit()

islower()

isnumeric()

isspace()

istitle()

isupper()

replace(*pat: str*, *repl: str*, *n: int = - 1*, *case: bool = True*, *regex: bool = True*)

Replace occurrences of pattern/regex in the Series/Index with some other string. Equivalent to str.replace() or re.sub().

Return string Series where in each row the occurrences of the given pattern or regex `pat` are replaced by `repl`.

This implementation differs to the one in `pandas`:

- We always return a missing for missing data.
- You cannot pass flags for the regular expression module.

Parameters

pat [str] Character sequence or regular expression.

repl [str] Replacement string.

n [int] Number of replacements to make from start.

case [bool, default True] If True, case sensitive.

regex [bool, default True] If True, assumes the pat is a regular expression. If False, treats the pat as a literal string.

Returns

Series of string values.

slice(*start=0*, *end=None*, *step=1*)
Extract every *step* character from strings from *start* to *end*.

startswith(*pat*)
Check whether a row starts with a certain pattern.

strip(*to_strip=None*)
Strip whitespaces from both ends of strings.

zfill(*width: int*) → pandas.core.series.Series
Pad strings in the Series/Index by prepending '0' characters.

fletcher.**pandas_from_arrow**(*arrow_object: Union[pyarrow.lib.RecordBatch, pyarrow.lib.Table, pyarrow.lib.Array, pyarrow.lib.ChunkedArray]*, *continuous: bool = False*)
Convert Arrow object instance to their Pandas equivalent by using Fletcher.

The conversion rules are:

- {RecordBatch, Table} -> DataFrame
- {Array, ChunkedArray} -> Series

Parameters

arrow_object [RecordBatch, Table, Array or ChunkedArray] object to be converted

continuous [bool] Use FletcherContinuousArray instead of FletcherChunkedArray

fletcher.**read_parquet**(*path*, *columns: Optional[List[str]] = None*, *continuous: bool = False*) → pandas.core.frame.DataFrame
Load a parquet object from the file path, returning a DataFrame with fletcher columns.

Parameters

path [str or file-like]

continuous [bool] Use FletcherContinuousArray instead of FletcherChunkedArray

Returns

pd.DataFrame

Use Apache Arrow backed columns in Pandas 0.23+ using the ExtensionArray interface.

Fletcher provides a generic implementation of the `ExtensionDtype` and `ExtensionArray` interfaces of Pandas for columns backed by Apache Arrow. By using it you can use any data type available in Apache Arrow natively in Pandas. Most prominently, `fletcher` provides native String und List types.

`fletcher` provides two, slightly different implementations. There is *`FletcherChunkedArray`* which is based on `pyarrow.ChunkedArray`, i.e. it consists of a collection of one or more continuous `pyarrow.Array` instances. Thus the backing memory can be a single memory region but it isn't required. This makes operations like `concat` copy-free as the result will be a `ChunkedArray` that consists of the union of the chunks of the inputs. In contrast it makes algorithm implementation a bit more complex as we need to implement all algorithms to iterate over all rows of all the arrays, not simply 0..n-1 of a single array.

The other implementation is *`FletcherContinuousArray`* which is based on a single `pyarrow.Array` instance. While this makes operations like `concat` more costly, it greatly improves usability and extensibility by being a much simpler structure. One can always assume that the backing memory region is a continuous block of memory and iterate with simple 0..n-1 indexing over the rows.

At the moment, we don't provide a default `FletcherArray`-named implementation as we are uncertain which of the two above implementations will be the most accepted one. Once we know to which implementation users converge, we will name that one `FletcherArray`.

In addition to bringing an alternative memory backend to NumPy, `fletcher` also provides high-performance operations on the new column types. It will either use the native implementation of an algorithm if provided in `pyarrow` or otherwise provide an implementation by itself using Numba.

Usage of fletcher columns is straightforward using Pandas' default constructor:

```
import fletcher as fr
import pandas as pd

df = pd.DataFrame({
    'str_chunked': fr.FletcherChunkedArray(['a', 'b', 'c']),
    'str_continuous': fr.FletcherContinuousArray(['a', 'b', 'c']),
})

df.info()

# <class 'pandas.core.frame.DataFrame'>
# RangeIndex: 3 entries, 0 to 2
# Data columns (total 2 columns):
#  #   Column          Non-Null Count  Dtype
# ---  ------          --------------  -----
#  0   str_chunked     3 non-null      fletcher_chunked[string]
#  1   str_continuous  3 non-null      fletcher_continuous[string]
# dtypes: fletcher_chunked[string](1), fletcher_continuous[string](1)
# memory usage: 166.0 bytes
```

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

fletcher, 25
fletcher.algorithms, 11
fletcher.algorithms.bool, 6
fletcher.algorithms.numpy_ufunc, 8
fletcher.algorithms.string, 8
fletcher.algorithms.string_builder, 8
fletcher.algorithms.string_builder_nojit, 9
fletcher.algorithms.utils, 6
fletcher.algorithms.utils.chunking, 5
fletcher.algorithms.utils.kmp, 6
fletcher.base, 11
fletcher.io, 21
fletcher.string_array, 21
fletcher.string_mixin, 24
fletcher.testing, 25

INDEX

A

all() (*fletcher.base.FletcherBaseArray method*), 12
all() (*fletcher.FletcherBaseArray method*), 26
all_op() (*in module fletcher.algorithms.bool*), 6
all_true() (*in module fletcher.algorithms.bool*), 6
all_true_like() (*in module fletcher.algorithms.bool*), 6
any() (*fletcher.base.FletcherBaseArray method*), 12
any() (*fletcher.FletcherBaseArray method*), 26
any_op() (*in module fletcher.algorithms.bool*), 6
append() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
append_bytes() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
append_false() (*fletcher.algorithms.string_builder_nojit.BitVector method*), 9
append_int16() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
append_int32() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
append_int64() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
append_null() (*fletcher.algorithms.string_builder_nojit.StringArrayBuilder method*), 10
append_to_kmp_matching() (*in module fletcher.algorithms.utils.kmp*), 6
append_true() (*fletcher.algorithms.string_builder_nojit.BitVector method*), 9
append_uint32() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
append_value() (*fletcher.algorithms.string_builder_nojit.StringArrayBuilder method*), 10
apply_binary_str() (*in module fletcher.algorithms.string*), 8
apply_per_chunk() (*in module fletcher.algorithms.utils.chunking*), 5
astype() (*fletcher.base.FletcherBaseArray method*), 12
astype() (*fletcher.FletcherBaseArray method*), 26

B

base() (*fletcher.base.FletcherBaseArray property*), 12
base() (*fletcher.FletcherBaseArray property*), 26
bitmap_or_unaligned() (*in module fletcher.algorithms.bool*), 7
bitmap_or_unaligned_with_numpy() (*in module fletcher.algorithms.bool*), 7
bitmap_or_unaligned_with_numpy_nonnull() (*in module fletcher.algorithms.bool*), 7
BitVector (*class in fletcher.algorithms.string_builder_nojit*), 9
buffers_as_arrays() (*in module fletcher.string_array*), 23
byte_for_bits() (*in module fletcher.algorithms.string_builder*), 8
byte_for_bits() (*in module fletcher.algorithms.string_builder_nojit*), 11
ByteVector (*class in fletcher.algorithms.string_builder_nojit*), 9

C

cat() (*fletcher.string_array.TextAccessor method*), 22
cat() (*fletcher.TextAccessor method*), 35
compute_kmp_failure_function() (*in module fletcher.algorithms.utils.kmp*), 6
construct_array_type() (*fletcher.base.FletcherChunkedDtype class method*), 17
construct_array_type() (*fletcher.base.FletcherContinuousDtype class method*), 20
construct_array_type() (*fletcher.FletcherChunkedDtype class method*), 31
construct_array_type() (*fletcher.FletcherContinuousDtype class method*), 34
construct_from_string() (*fletcher.base.FletcherChunkedDtype class method*), 17
construct_from_string()

(*fletcher.base.FletcherContinuousDtype class method*), 20
construct_from_string() (*fletcher.FletcherChunkedDtype class method*), 31
construct_from_string() (*fletcher.FletcherContinuousDtype class method*), 34
contains() (*fletcher.string_array.TextAccessor method*), 22
contains() (*fletcher.TextAccessor method*), 35
copy() (*fletcher.base.FletcherChunkedArray method*), 15
copy() (*fletcher.base.FletcherContinuousArray method*), 18
copy() (*fletcher.FletcherChunkedArray method*), 29
copy() (*fletcher.FletcherContinuousArray method*), 32
count() (*fletcher.string_array.TextAccessor method*), 22
count() (*fletcher.TextAccessor method*), 36

D

delete() (*fletcher.algorithms.string_builder_nojit.BitVector method*), 9
delete() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
delete() (*fletcher.algorithms.string_builder_nojit.StringArrayBuilder method*), 10
dispatch_chunked_binary_map() (*in module fletcher.algorithms.utils.chunking*), 5
dtype() (*fletcher.base.FletcherBaseArray property*), 12
dtype() (*fletcher.FletcherBaseArray property*), 26

E

endswith() (*fletcher.string_array.TextAccessor method*), 22
endswith() (*fletcher.TextAccessor method*), 36
example() (*fletcher.base.FletcherBaseDtype method*), 13
example() (*fletcher.FletcherBaseDtype method*), 27
examples() (*in module fletcher.testing*), 25
expand() (*fletcher.algorithms.string_builder_nojit.BitVector method*), 9
expand() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10

F

factorize() (*fletcher.base.FletcherChunkedArray method*), 15
factorize() (*fletcher.base.FletcherContinuousArray method*), 18
factorize() (*fletcher.FletcherChunkedArray method*), 29
factorize() (*fletcher.FletcherContinuousArray method*), 32
fillna() (*fletcher.base.FletcherChunkedArray method*), 15
fillna() (*fletcher.base.FletcherContinuousArray method*), 19
fillna() (*fletcher.FletcherChunkedArray method*), 29
fillna() (*fletcher.FletcherContinuousArray method*), 33
finalize_string_array() (*in module fletcher.algorithms.string_builder*), 8
finalize_string_array() (*in module fletcher.algorithms.string_builder_nojit*), 11
flatten() (*fletcher.base.FletcherChunkedArray method*), 16
flatten() (*fletcher.base.FletcherContinuousArray method*), 19
flatten() (*fletcher.FletcherChunkedArray method*), 30
flatten() (*fletcher.FletcherContinuousArray method*), 33
fletcher
    module, 25
fletcher.algorithms
    module, 11
fletcher.algorithms.bool
    module, 6
fletcher.algorithms.numpy_ufunc
    module, 8
fletcher.algorithms.string
    module, 8
fletcher.algorithms.string_builder
    module, 8
fletcher.algorithms.string_builder_nojit
    module, 9
fletcher.algorithms.utils
    module, 6
fletcher.algorithms.utils.chunking
    module, 5
fletcher.algorithms.utils.kmp
    module, 6
fletcher.base
    module, 11
fletcher.io
    module, 21
fletcher.string_array
    module, 21
fletcher.string_mixin
    module, 24
fletcher.testing
    module, 25
FletcherBaseArray (*class in fletcher*), 25
FletcherBaseArray (*class in fletcher.base*), 11

FletcherBaseDtype (*class in fletcher*), 27
FletcherBaseDtype (*class in fletcher.base*), 13
FletcherChunkedArray (*class in fletcher*), 28
FletcherChunkedArray (*class in fletcher.base*), 14
FletcherChunkedDtype (*class in fletcher*), 30
FletcherChunkedDtype (*class in fletcher.base*), 16
FletcherContinuousArray (*class in fletcher*), 31
FletcherContinuousArray (*class in fletcher.base*), 17
FletcherContinuousDtype (*class in fletcher*), 34
FletcherContinuousDtype (*class in fletcher.base*), 20

G

get() (*fletcher.algorithms.string_builder_nojit.BitVector method*), 9
get_int16() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
get_int32() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
get_int64() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
get_uint32() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
get_uint8() (*fletcher.algorithms.string_builder_nojit.ByteVector method*), 10
get_utf8_size() (*in module fletcher.algorithms.string*), 8

I

isalnum() (*fletcher.string_array.TextAccessor method*), 22
isalnum() (*fletcher.TextAccessor method*), 36
isalpha() (*fletcher.string_array.TextAccessor method*), 22
isalpha() (*fletcher.TextAccessor method*), 36
isdecimal() (*fletcher.string_array.TextAccessor method*), 23
isdecimal() (*fletcher.TextAccessor method*), 36
isdigit() (*fletcher.string_array.TextAccessor method*), 23
isdigit() (*fletcher.TextAccessor method*), 36
islower() (*fletcher.string_array.TextAccessor method*), 23
islower() (*fletcher.TextAccessor method*), 36
isna() (*fletcher.base.FletcherBaseArray method*), 12
isna() (*fletcher.FletcherBaseArray method*), 26
isnumeric() (*fletcher.string_array.TextAccessor method*), 23
isnumeric() (*fletcher.TextAccessor method*), 36
isspace() (*fletcher.string_array.TextAccessor method*), 23
isspace() (*fletcher.TextAccessor method*), 36
istitle() (*fletcher.string_array.TextAccessor method*), 23
istitle() (*fletcher.TextAccessor method*), 36
isupper() (*fletcher.string_array.TextAccessor method*), 23
isupper() (*fletcher.TextAccessor method*), 36
itemsize() (*fletcher.base.FletcherBaseDtype property*), 13
itemsize() (*fletcher.FletcherBaseDtype property*), 27

K

kind() (*fletcher.base.FletcherBaseDtype property*), 13
kind() (*fletcher.FletcherBaseDtype property*), 27

L

LibcMalloc (*class in fletcher.algorithms.string_builder*), 8

M

malloc_nojit() (*in module fletcher.algorithms.string_builder*), 8
masked_bitmap_or_unaligned() (*in module fletcher.algorithms.bool*), 7
module
    fletcher, 25
    fletcher.algorithms, 11
    fletcher.algorithms.bool, 6
    fletcher.algorithms.numpy_ufunc, 8
    fletcher.algorithms.string, 8
    fletcher.algorithms.string_builder, 8
    fletcher.algorithms.string_builder_nojit, 9
    fletcher.algorithms.utils, 6
    fletcher.algorithms.utils.chunking, 5
    fletcher.algorithms.utils.kmp, 6
    fletcher.base, 11
    fletcher.io, 21
    fletcher.string_array, 21
    fletcher.string_mixin, 24
    fletcher.testing, 25

N

na_value (*fletcher.base.FletcherBaseDtype attribute*), 13
na_value (*fletcher.FletcherBaseDtype attribute*), 27
name() (*fletcher.base.FletcherBaseDtype property*), 13
name() (*fletcher.FletcherBaseDtype property*), 27
nbytes() (*fletcher.base.FletcherChunkedArray property*), 16
nbytes() (*fletcher.base.FletcherContinuousArray property*), 19

nbytes() (*fletcher.FletcherChunkedArray property*), 30
nbytes() (*fletcher.FletcherContinuousArray property*), 33
ndim() (*fletcher.base.FletcherBaseArray property*), 12
ndim() (*fletcher.FletcherBaseArray property*), 26

O

or_array_array() (*in module fletcher.algorithms.bool*), 7
or_array_nparray() (*in module fletcher.algorithms.bool*), 7
or_na() (*in module fletcher.algorithms.bool*), 7
or_vectorised() (*in module fletcher.algorithms.bool*), 7

P

pandas_from_arrow() (*in module fletcher*), 37
pandas_from_arrow() (*in module fletcher.base*), 21

R

read_parquet() (*in module fletcher*), 37
read_parquet() (*in module fletcher.io*), 21
replace() (*fletcher.string_array.TextAccessor method*), 23
replace() (*fletcher.TextAccessor method*), 36

S

shape() (*fletcher.base.FletcherBaseArray property*), 12
shape() (*fletcher.FletcherBaseArray property*), 26
shift_unaligned_bitmap() (*in module fletcher.algorithms.string*), 8
signature() (*fletcher.algorithms.string_builder.LibcMalloc method*), 8
size() (*fletcher.base.FletcherBaseArray property*), 12
size() (*fletcher.FletcherBaseArray property*), 26
slice() (*fletcher.string_array.TextAccessor method*), 23
slice() (*fletcher.TextAccessor method*), 37
startswith() (*fletcher.string_array.TextAccessor method*), 23
startswith() (*fletcher.TextAccessor method*), 37
StringArrayBuilder (*class in fletcher.algorithms.string_builder_nojit*), 10
StringSupportingExtensionArray (*class in fletcher.string_mixin*), 24
strip() (*fletcher.string_array.TextAccessor method*), 23
strip() (*fletcher.TextAccessor method*), 37
sum() (*fletcher.base.FletcherBaseArray method*), 12
sum() (*fletcher.FletcherBaseArray method*), 26

T

take() (*fletcher.base.FletcherChunkedArray method*), 16
take() (*fletcher.base.FletcherContinuousArray method*), 19
take() (*fletcher.FletcherChunkedArray method*), 30
take() (*fletcher.FletcherContinuousArray method*), 33
TextAccessor (*class in fletcher*), 35
TextAccessor (*class in fletcher.string_array*), 21
TextAccessorBase (*class in fletcher.string_array*), 23
TextAccessorExt (*class in fletcher.string_array*), 23
type() (*fletcher.base.FletcherBaseDtype property*), 14
type() (*fletcher.FletcherBaseDtype property*), 28

U

unique() (*fletcher.base.FletcherBaseArray method*), 12
unique() (*fletcher.FletcherBaseArray method*), 26

V

value_counts() (*fletcher.base.FletcherBaseArray method*), 13
value_counts() (*fletcher.FletcherBaseArray method*), 27

Z

zfill() (*fletcher.string_array.TextAccessor method*), 23
zfill() (*fletcher.TextAccessor method*), 37