pandas.DataFrame.astype

`DataFrame.astype(dtype, copy=None, errors='raise')` [source]

Cast a pandas object to a specified dtype `dtype`.

Parameters:

dtype : *str, data type, Series or Mapping of column name -> data type*

Use a str, numpy.dtype, pandas.ExtensionDtype or Python type to cast entire pandas object to the same type. Alternatively, use a mapping, e.g. {col: dtype, ...}, where col is a column label and dtype is a numpy.dtype or Python type to cast one or more of the DataFrame's columns to column-specific types.

copy : *bool, default True*

Return a copy when `copy=True` (be very careful setting `copy=False` as changes to values then may propagate to other pandas objects).

> **Note**
>
> The *copy* keyword will change behavior in pandas 3.0. Copy-on-Write will be enabled by default, which means that all methods with a *copy* keyword will use a lazy copy mechanism to defer the copy and ignore the *copy* keyword. The *copy* keyword will be removed in a future version of pandas.
> You can already get the future behavior and improvements through enabling copy on write `pd.options.mode.copy_on_write = True`

errors : *{'raise', 'ignore'}, default 'raise'*

Control raising of exceptions on invalid data for provided dtype.

- `raise` : allow exceptions to be raised
- `ignore` : suppress exceptions. On error return original object.

Returns:

Skip to main content

See also

`to_datetime`
: Convert argument to datetime.

`to_timedelta`
: Convert argument to timedelta.

`to_numeric`
: Convert argument to a numeric type.

`numpy.ndarray.astype`
: Cast a numpy array to a specified type.

Notes

Changed in version 2.0.0: Using `astype` to convert from timezone-naive dtype to timezone-aware dtype will raise an exception. Use `Series.dt.tz_localize()` instead.

Examples

Create a DataFrame:

```
>>> d = {'col1': [1, 2], 'col2': [3, 4]}
>>> df = pd.DataFrame(data=d)
>>> df.dtypes
col1    int64
col2    int64
dtype: object
```

Cast all columns to int32:

```
>>> df.astype('int32').dtypes
col1    int32
col2    int32
dtype: object
```

Cast col1 to int32 using a dictionary:

```
>>> df.astype({'col1': 'int32'}).dtypes
```

```
col2    int64
dtype: object
```

Create a series:

```
>>> ser = pd.Series([1, 2], dtype='int32')
>>> ser
0    1
1    2
dtype: int32
>>> ser.astype('int64')
0    1
1    2
dtype: int64
```

Convert to categorical type:

```
>>> ser.astype('category')
0    1
1    2
dtype: category
Categories (2, int32): [1, 2]
```

Convert to ordered categorical type with custom ordering:

```
>>> from pandas.api.types import CategoricalDtype
>>> cat_dtype = CategoricalDtype(
...     categories=[2, 1], ordered=True)
>>> ser.astype(cat_dtype)
0    1
1    2
dtype: category
Categories (2, int64): [2 < 1]
```

Create a series of dates:

```
>>> ser_date = pd.Series(pd.date_range('20200101', periods=3))
>>> ser_date
0   2020-01-01
1   2020-01-02
2   2020-01-03
dtype: datetime64[ns]
```

Previous
pandas.DataFrame.set_flags

Next
pandas.DataFrame.convert_dtypes