Input/output

Pickling

- **read_pickle (filepath_or_buffer[, ...])** - Load pickled pandas object (or any object) from file.

- **DataFrame.to_pickle (path, *[, compression, ...])** - Pickle (serialize) object to file.

Flat file

- **read_table (filepath_or_buffer, *[, sep, ...])** - Read general delimited file into DataFrame.

- **read_csv (filepath_or_buffer, *[, sep, ...])** - Read a comma-separated values (csv) file into DataFrame.
- **DataFrame.to_csv ([path_or_buf, sep, na_rep, ...])** - Write object to a comma-separated values (csv) file.

- **read_fwf (filepath_or_buffer, *[, colspecs, ...])** - Read a table of fixed-width formatted lines into DataFrame.

Clipboard

- **read_clipboard ([sep, dtype_backend])** - Read text from clipboard and pass to read_csv() .

- **DataFrame.to_clipboard (*[, excel, sep])** - Copy object to the system clipboard.

Excel

- **read_excel (io[, sheet_name, header, names, ...])** - Read an Excel file into a pandas DataFrame .

- **DataFrame.to_excel (excel_writer, *[, ...])** - Write object to an Excel sheet.
- **ExcelFile (path_or_buffer[, engine, ...])** - Class for parsing tabular Excel sheets into DataFrame objects.
- **ExcelFile.book** -
- **ExcelFile.sheet_names** -
- **ExcelFile.parse ([sheet_name, header, names, ...])** - Parse specified sheet(s) into a DataFrame.
- **Styler.to_excel (excel_writer[, sheet_name, ...])** - Write Styler to an Excel sheet.

- **ExcelWriter (path[, engine, date_format, ...])** - Class for writing DataFrame objects into excel sheets.

JSON

- **read_json (path_or_buf, *[, orient, typ, ...])** - Convert a JSON string to pandas object.
- **json_normalize (data[, record_path, meta, ...])** - Normalize semi-structured JSON data into a flat table.
- **DataFrame.to_json ([path_or_buf, orient, ...])** - Convert the object to a JSON string.
- **build_table_schema (data[, index, ...])** - Create a Table schema from data .

HTML

- **read_html (io, *[, match, flavor, header, ...])** - Read HTML tables into a list of DataFrame objects.
- **DataFrame.to_html ([buf, columns, col_space, ...])** - Render a DataFrame as an HTML table.
- **Styler.to_html ([buf, table_uuid, ...])** - Write Styler to a file, buffer or string in HTML-CSS format.

XML

- **read_xml (path_or_buffer, *[, xpath, ...])** - Read XML document into a DataFrame object.
- **DataFrame.to_xml ([path_or_buffer, index, ...])** - Render a DataFrame to an XML document.

Latex

- **DataFrame.to_latex ([buf, columns, header, ...])** - Render object to a LaTeX tabular, longtable, or nested table.
- **Styler.to_latex ([buf, column_format, ...])** - Write Styler to a file, buffer or string in LaTeX format.

HDFStore: PyTables (HDF5)

- **read_hdf (path_or_buf[, key, mode, errors, ...])** - Read from the store, close it if we opened it.
- **HDFStore.put (key, value[, format, index, ...])** - Store object in HDFStore.
- **HDFStore.append (key, value[, format, axes, ...])** - Append to Table in file.
- **HDFStore.get (key)** - Retrieve pandas object stored in file.

- **HDFStore.select (key[, where, start, stop, ...])** - Retrieve pandas object stored in file, optionally based on where criteria.
- **HDFStore.info ()** - Print detailed information on the store.
- **HDFStore.keys ([include])** - Return a list of keys corresponding to objects stored in HDFStore.
- **HDFStore.groups ()** - Return a list of all the top-level nodes.

- **HDFStore.walk ([where])** - Walk the pytables group hierarchy for pandas objects.

Feather

- **read_feather (path[, columns, use_threads, ...])** - Load a feather-format object from the file path.

- **DataFrame.to_feather (path, **kwargs)** - Write a DataFrame to the binary Feather format.

Parquet

- **read_parquet (path[, engine, columns, ...])** - Load a parquet object from the file path, returning a DataFrame.

- **DataFrame.to_parquet ([path, engine, ...])** - Write a DataFrame to the binary parquet format.

ORC

- **read_orc (path[, columns, dtype_backend, ...])** - Load an ORC object from the file path, returning a DataFrame.

- **DataFrame.to_orc ([path, engine, index, ...])** - Write a DataFrame to the ORC format.

SAS

- **read_sas (filepath_or_buffer, *[, format, ...])** - Read SAS files stored as either XPORT or SAS7BDAT format files.

SPSS

- **read_spss (path[, usecols, ...])** - Load an SPSS file from the file path, returning a DataFrame.

SQL

- **read_sql_table (table_name, con[, schema, ...])** - Read SQL database table into a DataFrame.

- **read_sql_query (sql, con[, index_col, ...])** - Read SQL query into a DataFrame.
- **read_sql (sql, con[, index_col, ...])** - Read SQL query or database table into a DataFrame.

- **DataFrame.to_sql (name, con, *[, schema, ...])** - Write records stored in a DataFrame to a SQL database.

Google BigQuery

- **read_gbq (query[, project_id, index_col, ...])** - (DEPRECATED) Load data from Google BigQuery.

STATA

- **read_stata (filepath_or_buffer, *[, ...])** - Read Stata file into DataFrame.

- **DataFrame.to_stata (path, *[, convert_dates, ...])** - Export DataFrame object to Stata dta format.
- **StataReader.data_label** - Return data label of Stata file.
- **StataReader.value_labels ()** - Return a nested dict associating each variable name to its value and label.
- **StataReader.variable_labels ()** - Return a dict associating each variable name with corresponding label.
- **StataWriter.write_file ()** - Export DataFrame object to Stata dta format.

General functions

Data manipulations

- **melt (frame[, id_vars, value_vars, var_name, ...])** - Unpivot a DataFrame from wide to long format, optionally leaving identifiers set.
- **pivot (data, *, columns[, index, values])** - Return reshaped DataFrame organized by given index / column values.
- **pivot_table (data[, values, index, columns, ...])** - Create a spreadsheet-style pivot table as a DataFrame.
- **crosstab (index, columns[, values, rownames, ...])** - Compute a simple cross tabulation of two (or more) factors.
- **cut (x, bins[, right, labels, retbins, ...])** - Bin values into discrete intervals.
- **qcut (x, q[, labels, retbins, precision, ...])** - Quantile-based discretization function.

- **merge (left, right[, how, on, left_on, ...])** - Merge DataFrame or named Series objects with a database-style join.
- **merge_ordered (left, right[, on, left_on, ...])** - Perform a merge for ordered data with optional filling/interpolation.
- **merge_asof (left, right[, on, left_on, ...])** - Perform a merge by key distance.
- **concat (objs, *[, axis, join, ignore_index, ...])** - Concatenate pandas objects along a particular axis.
- **get_dummies (data[, prefix, prefix_sep, ...])** - Convert categorical variable into dummy/indicator variables.
- **from_dummies (data[, sep, default_category])** - Create a categorical DataFrame from a DataFrame of dummy variables.
- **factorize (values[, sort, use_na_sentinel, ...])** - Encode the object as an enumerated type or categorical variable.
- **unique (values)** - Return unique values based on a hash table.
- **lreshape (data, groups[, dropna])** - Reshape wide-format data to long.

- **wide_to_long (df, stubnames, i, j[, sep, suffix])** - Unpivot a DataFrame from wide to long format.

Top-level missing data

- **isna (obj)** - Detect missing values for an array-like object.

- **isnull (obj)** - Detect missing values for an array-like object.
- **notna (obj)** - Detect non-missing values for an array-like object.

- **notnull (obj)** - Detect non-missing values for an array-like object.

Top-level dealing with numeric data

- **to_numeric (arg[, errors, downcast, ...])** - Convert argument to a numeric type.

Top-level dealing with datetimelike data

- **to_datetime (arg[, errors, dayfirst, ...])** - Convert argument to datetime.

- **to_timedelta (arg[, unit, errors])** - Convert argument to timedelta.
- **date_range ([start, end, periods, freq, tz, ...])** - Return a fixed frequency DatetimeIndex.
- **bdate_range ([start, end, periods, freq, tz, ...])** - Return a fixed frequency DatetimeIndex with business day as the default.
- **period_range ([start, end, periods, freq, name])** - Return a fixed frequency PeriodIndex.

- **timedelta_range ([start, end, periods, freq, ...])** - Return a fixed frequency TimedeltaIndex with day as the default.
- **infer_freq (index)** - Infer the most likely frequency given the input index.

Top-level dealing with Interval data

- **interval_range ([start, end, periods, freq, ...])** - Return a fixed frequency IntervalIndex.

Top-level evaluation

- **eval (expr[, parser, engine, local_dict, ...])** - Evaluate a Python expression as a string using various backends.

Datetime formats

- **tseries.api.guess_datetime_format (dt_str[, ...])** - Guess the datetime format of a given datetime string.

Hashing

- **util.hash_array (vals[, encoding, hash_key, ...])** - Given a 1d array, return an array of deterministic integers.
- **util.hash_pandas_object (obj[, index, ...])** - Return a data hash of the Index/Series/DataFrame.

Importing from other DataFrame libraries

- **api.interchange.from_dataframe (df[, allow_copy])** - Build a pd.DataFrame from any DataFrame supporting the interchange protocol.

Series

Constructor

- **Series ([data, index, dtype, name, copy, ...])** - One-dimensional ndarray with axis labels (including time series).

Attributes

- **Series.index** - The index (axis labels) of the Series.

- **Series.array** - The ExtensionArray of the data backing this Series or Index.
- **Series.values** - Return Series as ndarray or ndarray-like depending on the dtype.
- **Series.dtype** - Return the dtype object of the underlying data.
- **Series.shape** - Return a tuple of the shape of the underlying data.
- **Series.nbytes** - Return the number of bytes in the underlying data.
- **Series.ndim** - Number of dimensions of the underlying data, by definition 1.
- **Series.size** - Return the number of elements in the underlying data.
- **Series.T** - Return the transpose, which is by definition self.
- **Series.memory_usage ([index, deep])** - Return the memory usage of the Series.
- **Series.hasnans** - Return True if there are any NaNs.
- **Series.empty** - Indicator whether Series/DataFrame is empty.
- **Series.dtypes** - Return the dtype object of the underlying data.
- **Series.name** - Return the name of the Series.
- **Series.flags** - Get the properties associated with this pandas object.

- **Series.set_flags (*[, copy, ...])** - Return a new object with updated flags.

Conversion

- **Series.astype (dtype[, copy, errors])** - Cast a pandas object to a specified dtype dtype .

- **Series.convert_dtypes ([infer_objects, ...])** - Convert columns to the best possible dtypes using dtypes supporting pd.NA .
- **Series.infer_objects ([copy])** - Attempt to infer better dtypes for object columns.
- **Series.copy ([deep])** - Make a copy of this object's indices and data.
- **Series.bool ()** - (DEPRECATED) Return the bool of a single element Series or DataFrame.
- **Series.to_numpy ([dtype, copy, na_value])** - A NumPy ndarray representing the values in this Series or Index.
- **Series.to_period ([freq, copy])** - Convert Series from DatetimeIndex to PeriodIndex.
- **Series.to_timestamp ([freq, how, copy])** - Cast to DatetimeIndex of Timestamps, at beginning of period.
- **Series.to_list ()** - Return a list of the values.

- **Series.__array__ ([dtype, copy])** - Return the values as a NumPy array.

Indexing, iteration

- **Series.get (key[, default])** - Get item from object for given key (ex: DataFrame column).

- **Series.at** - Access a single value for a row/column label pair.
- **Series.iat** - Access a single value for a row/column pair by integer position.
- **Series.loc** - Access a group of rows and columns by label(s) or a boolean array.
- **Series.iloc** - (DEPRECATED) Purely integer-location based indexing for selection by position.
- **Series.__iter__ ()** - Return an iterator of the values.
- **Series.items ()** - Lazily iterate over (index, value) tuples.
- **Series.keys ()** - Return alias for index.
- **Series.pop (item)** - Return item and drops from series.
- **Series.item ()** - Return the first element of the underlying data as a Python scalar.

- **Series.xs (key[, axis, level, drop_level])** - Return cross-section from the Series/DataFrame.

Binary operator functions

- **Series.add (other[, level, fill_value, axis])** - Return Addition of series and other, element-wise (binary operator add).

- **Series.sub (other[, level, fill_value, axis])** - Return Subtraction of series and other, element-wise (binary operator sub).
- **Series.mul (other[, level, fill_value, axis])** - Return Multiplication of series and other, element-wise (binary operator mul).
- **Series.div (other[, level, fill_value, axis])** - Return Floating division of series and other, element-wise (binary operator truediv).
- **Series.truediv (other[, level, fill_value, axis])** - Return Floating division of series and other, element-wise (binary operator truediv).
- **Series.floordiv (other[, level, fill_value, axis])** - Return Integer division of series and other, element-wise (binary operator floordiv).
- **Series.mod (other[, level, fill_value, axis])** - Return Modulo of series and other, element-wise (binary operator mod).
- **Series.pow (other[, level, fill_value, axis])** - Return Exponential power of series and other, element-wise (binary operator pow).
- **Series.radd (other[, level, fill_value, axis])** - Return Addition of series and other, element-wise (binary operator radd).

- **Series.rsub (other[, level, fill_value, axis])** - Return Subtraction of series and other, element-wise (binary operator rsub).
- **Series.rmul (other[, level, fill_value, axis])** - Return Multiplication of series and other, element-wise (binary operator rmul).
- **Series.rdiv (other[, level, fill_value, axis])** - Return Floating division of series and other, element-wise (binary operator rtruediv).
- **Series.rtruediv (other[, level, fill_value, axis])** - Return Floating division of series and other, element-wise (binary operator rtruediv).
- **Series.rfloordiv (other[, level, fill_value, ...])** - Return Integer division of series and other, element-wise (binary operator rfloordiv).
- **Series.rmod (other[, level, fill_value, axis])** - Return Modulo of series and other, element-wise (binary operator rmod).
- **Series.rpow (other[, level, fill_value, axis])** - Return Exponential power of series and other, element-wise (binary operator rpow).
- **Series.combine (other, func[, fill_value])** - Combine the Series with a Series or scalar according to func .
- **Series.combine_first (other)** - Update null elements with value in the same location in 'other'.
- **Series.round ([decimals])** - Round each value in a Series to the given number of decimals.
- **Series.lt (other[, level, fill_value, axis])** - Return Less than of series and other, element-wise (binary operator lt).
- **Series.gt (other[, level, fill_value, axis])** - Return Greater than of series and other, element-wise (binary operator gt).
- **Series.le (other[, level, fill_value, axis])** - Return Less than or equal to of series and other, element-wise (binary operator le).
- **Series.ge (other[, level, fill_value, axis])** - Return Greater than or equal to of series and other, element-wise (binary operator ge).
- **Series.ne (other[, level, fill_value, axis])** - Return Not equal to of series and other, element-wise (binary operator ne).
- **Series.eq (other[, level, fill_value, axis])** - Return Equal to of series and other, element-wise (binary operator eq).
- **Series.product ([axis, skipna, numeric_only, ...])** - Return the product of the values over the requested axis.

- **Series.dot (other)** - Compute the dot product between the Series and the columns of other.

Function application, GroupBy & window

- **Series.apply (func[, convert_dtype, args, by_row])** - Invoke function on values of Series.

- **Series.agg ([func, axis])** - Aggregate using one or more operations over the specified axis.
- **Series.aggregate ([func, axis])** - Aggregate using one or more operations over the specified axis.
- **Series.transform (func[, axis])** - Call func on self producing a Series with the same axis shape as self.
- **Series.map (arg[, na_action])** - Map values of Series according to an input mapping or function.
- **Series.groupby ([by, axis, level, as_index, ...])** - Group Series using a mapper or by a Series of columns.
- **Series.rolling (window[, min_periods, ...])** - Provide rolling window calculations.
- **Series.expanding ([min_periods, axis, method])** - Provide expanding window calculations.
- **Series.ewm ([com, span, halflife, alpha, ...])** - Provide exponentially weighted (EW) calculations.

- **Series.pipe (func, *args, **kwargs)** - Apply chainable functions that expect Series or DataFrames.

Computations / descriptive stats

- **Series.abs ()** - Return a Series/DataFrame with absolute numeric value of each element.

- **Series.all ([axis, bool_only, skipna])** - Return whether all elements are True, potentially over an axis.
- **Series.any (*[, axis, bool_only, skipna])** - Return whether any element is True, potentially over an axis.
- **Series.autocorr ([lag])** - Compute the lag-N autocorrelation.
- **Series.between (left, right[, inclusive])** - Return boolean Series equivalent to left <\= series <\= right.
- **Series.clip ([lower, upper, axis, inplace])** - Trim values at input threshold(s).
- **Series.corr (other[, method, min_periods])** - Compute correlation with other Series, excluding missing values.
- **Series.count ()** - Return number of non-NA/null observations in the Series.
- **Series.cov (other[, min_periods, ddof])** - Compute covariance with Series, excluding missing values.
- **Series.cummax ([axis, skipna])** - Return cumulative maximum over a DataFrame or Series axis.
- **Series.cummin ([axis, skipna])** - Return cumulative minimum over a DataFrame or Series axis.
- **Series.cumprod ([axis, skipna])** - Return cumulative product over a DataFrame or Series axis.
- **Series.cumsum ([axis, skipna])** - Return cumulative sum over a DataFrame or Series axis.
- **Series.describe ([percentiles, include, exclude])** - Generate descriptive statistics.
- **Series.diff ([periods])** - First discrete difference of element.

- **Series.factorize ([sort, use_na_sentinel])** - Encode the object as an enumerated type or categorical variable.
- **Series.kurt ([axis, skipna, numeric_only])** - Return unbiased kurtosis over requested axis.
- **Series.max ([axis, skipna, numeric_only])** - Return the maximum of the values over the requested axis.
- **Series.mean ([axis, skipna, numeric_only])** - Return the mean of the values over the requested axis.
- **Series.median ([axis, skipna, numeric_only])** - Return the median of the values over the requested axis.
- **Series.min ([axis, skipna, numeric_only])** - Return the minimum of the values over the requested axis.
- **Series.mode ([dropna])** - Return the mode(s) of the Series.
- **Series.nlargest ([n, keep])** - Return the largest n elements.
- **Series.nsmallest ([n, keep])** - Return the smallest n elements.
- **Series.pct_change ([periods, fill_method, ...])** - Fractional change between the current and a prior element.
- **Series.prod ([axis, skipna, numeric_only, ...])** - Return the product of the values over the requested axis.
- **Series.quantile ([q, interpolation])** - Return value at the given quantile.
- **Series.rank ([axis, method, numeric_only, ...])** - Compute numerical data ranks (1 through n) along axis.
- **Series.sem ([axis, skipna, ddof, numeric_only])** - Return unbiased standard error of the mean over requested axis.
- **Series.skew ([axis, skipna, numeric_only])** - Return unbiased skew over requested axis.
- **Series.std ([axis, skipna, ddof, numeric_only])** - Return sample standard deviation over requested axis.
- **Series.sum ([axis, skipna, numeric_only, ...])** - Return the sum of the values over the requested axis.
- **Series.var ([axis, skipna, ddof, numeric_only])** - Return unbiased variance over requested axis.
- **Series.kurtosis ([axis, skipna, numeric_only])** - Return unbiased kurtosis over requested axis.
- **Series.unique ()** - Return unique values of Series object.
- **Series.nunique ([dropna])** - Return number of unique elements in the object.
- **Series.is_unique** - Return boolean if values in the object are unique.
- **Series.is_monotonic_increasing** - Return boolean if values in the object are monotonically increasing.
- **Series.is_monotonic_decreasing** - Return boolean if values in the object are monotonically decreasing.

- **Series.value_counts ([normalize, sort, ...])** - Return a Series containing counts of unique values.

Reindexing / selection / label manipulation

- **Series.align (other[, join, axis, level, ...])** - Align two objects on their axes with the specified join method.

- **Series.case_when (caselist)** - Replace values where the conditions are True.
- **Series.drop ([labels, axis, index, columns, ...])** - Return Series with specified index labels removed.
- **Series.droplevel (level[, axis])** - Return Series/DataFrame with requested index / column level(s) removed.
- **Series.drop_duplicates (*[, keep, inplace, ...])** - Return Series with duplicate values removed.
- **Series.duplicated ([keep])** - Indicate duplicate Series values.
- **Series.equals (other)** - Test whether two objects contain the same elements.
- **Series.first (offset)** - (DEPRECATED) Select initial periods of time series data based on a date offset.
- **Series.head ([n])** - Return the first n rows.
- **Series.idxmax ([axis, skipna])** - Return the row label of the maximum value.
- **Series.idxmin ([axis, skipna])** - Return the row label of the minimum value.
- **Series.isin (values)** - Whether elements in Series are contained in values .
- **Series.last (offset)** - (DEPRECATED) Select final periods of time series data based on a date offset.
- **Series.reindex ([index, axis, method, copy, ...])** - Conform Series to new index with optional filling logic.
- **Series.reindex_like (other[, method, copy, ...])** - Return an object with matching indices as other object.
- **Series.rename ([index, axis, copy, inplace, ...])** - Alter Series index labels or name.
- **Series.rename_axis ([mapper, index, axis, ...])** - Set the name of the axis for the index or columns.
- **Series.reset_index ([level, drop, name, ...])** - Generate a new DataFrame or Series with the index reset.
- **Series.sample ([n, frac, replace, weights, ...])** - Return a random sample of items from an axis of object.
- **Series.set_axis (labels, *[, axis, copy])** - Assign desired index to given axis.
- **Series.take (indices[, axis])** - Return the elements in the given positional indices along an axis.
- **Series.tail ([n])** - Return the last n rows.
- **Series.truncate ([before, after, axis, copy])** - Truncate a Series or DataFrame before and after some index value.
- **Series.where (cond[, other, inplace, axis, level])** - Replace values where the condition is False.
- **Series.mask (cond[, other, inplace, axis, level])** - Replace values where the condition is True.

- **Series.add_prefix (prefix[, axis])** - Prefix labels with string prefix .
- **Series.add_suffix (suffix[, axis])** - Suffix labels with string suffix .

- **Series.filter ([items, like, regex, axis])** - Subset the dataframe rows or columns according to the specified index labels.

Missing data handling

- **Series.backfill (*[, axis, inplace, limit, ...])** - (DEPRECATED) Fill NA/NaN values by using the next valid observation to fill the gap.

- **Series.bfill (*[, axis, inplace, limit, ...])** - Fill NA/NaN values by using the next valid observation to fill the gap.
- **Series.dropna (*[, axis, inplace, how, ...])** - Return a new Series with missing values removed.
- **Series.ffill (*[, axis, inplace, limit, ...])** - Fill NA/NaN values by propagating the last valid observation to next valid.
- **Series.fillna ([value, method, axis, ...])** - Fill NA/NaN values using the specified method.
- **Series.interpolate ([method, axis, limit, ...])** - Fill NaN values using an interpolation method.
- **Series.isna ()** - Detect missing values.
- **Series.isnull ()** - Series.isnull is an alias for Series.isna.
- **Series.notna ()** - Detect existing (non-missing) values.
- **Series.notnull ()** - Series.notnull is an alias for Series.notna.
- **Series.pad (*[, axis, inplace, limit, downcast])** - (DEPRECATED) Fill NA/NaN values by propagating the last valid observation to next valid.

- **Series.replace ([to_replace, value, inplace, ...])** - Replace values given in to_replace with value .

Reshaping, sorting

- **Series.argsort ([axis, kind, order, stable])** - Return the integer indices that would sort the Series values.

- **Series.argmin ([axis, skipna])** - Return int position of the smallest value in the Series.
- **Series.argmax ([axis, skipna])** - Return int position of the largest value in the Series.
- **Series.reorder_levels (order)** - Rearrange index levels using input order.
- **Series.sort_values (*[, axis, ascending, ...])** - Sort by the values.

- **Series.sort_index (*[, axis, level, ...])** - Sort Series by index labels.
- **Series.swaplevel ([i, j, copy])** - Swap levels i and j in a MultiIndex .
- **Series.unstack ([level, fill_value, sort])** - Unstack, also known as pivot, Series with MultiIndex to produce DataFrame.
- **Series.explode ([ignore_index])** - Transform each element of a list-like to a row.
- **Series.searchsorted (value[, side, sorter])** - Find indices where elements should be inserted to maintain order.
- **Series.ravel ([order])** - (DEPRECATED) Return the flattened underlying data as an ndarray or ExtensionArray.
- **Series.repeat (repeats[, axis])** - Repeat elements of a Series.
- **Series.squeeze ([axis])** - Squeeze 1 dimensional axis objects into scalars.

- **Series.view ([dtype])** - (DEPRECATED) Create a new view of the Series.

Combining / comparing / joining / merging

- **Series.compare (other[, align_axis, ...])** - Compare to another Series and show the differences.

- **Series.update (other)** - Modify Series in place using values from passed Series.

Time Series-related

- **Series.asfreq (freq[, method, how, ...])** - Convert time series to specified frequency.

- **Series.asof (where[, subset])** - Return the last row(s) without any NaNs before where .
- **Series.shift ([periods, freq, axis, ...])** - Shift index by desired number of periods with an optional time freq .
- **Series.first_valid_index ()** - Return index for first non-NA value or None, if no non-NA value is found.
- **Series.last_valid_index ()** - Return index for last non-NA value or None, if no non-NA value is found.
- **Series.resample (rule[, axis, closed, label, ...])** - Resample time-series data.
- **Series.tz_convert (tz[, axis, level, copy])** - Convert tz-aware axis to target time zone.
- **Series.tz_localize (tz[, axis, level, copy, ...])** - Localize tz-naive index of a Series or DataFrame to target time zone.
- **Series.at_time (time[, asof, axis])** - Select values at particular time of day (e.g., 9:30AM).

- **Series.between_time (start_time, end_time[, ...])** - Select values between particular times of the day (e.g., 9:00-9:30 AM).

Accessors

- **Series.str** - alias of StringMethods

- **Series.cat** - alias of CategoricalAccessor
- **Series.dt** - alias of CombinedDatetimelikeProperties
- **Series.sparse** - alias of SparseAccessor
- **DataFrame.sparse** - alias of SparseFrameAccessor
- **Index.str** - alias of StringMethods
- **Datetime, Timedelta, Period** - dt
- **String** - str
- **Categorical** - cat
- **Sparse** - sparse
- **Series.dt.date** - Returns numpy array of python datetime.date objects.
- **Series.dt.time** - Returns numpy array of datetime.time objects.
- **Series.dt.timetz** - Returns numpy array of datetime.time objects with timezones.
- **Series.dt.year** - The year of the datetime.
- **Series.dt.month** - The month as January\=1, December\=12.
- **Series.dt.day** - The day of the datetime.
- **Series.dt.hour** - The hours of the datetime.
- **Series.dt.minute** - The minutes of the datetime.
- **Series.dt.second** - The seconds of the datetime.
- **Series.dt.microsecond** - The microseconds of the datetime.
- **Series.dt.nanosecond** - The nanoseconds of the datetime.
- **Series.dt.dayofweek** - The day of the week with Monday\=0, Sunday\=6.
- **Series.dt.day_of_week** - The day of the week with Monday\=0, Sunday\=6.
- **Series.dt.weekday** - The day of the week with Monday\=0, Sunday\=6.
- **Series.dt.dayofyear** - The ordinal day of the year.
- **Series.dt.day_of_year** - The ordinal day of the year.
- **Series.dt.days_in_month** - The number of days in the month.
- **Series.dt.quarter** - The quarter of the date.
- **Series.dt.is_month_start** - Indicates whether the date is the first day of the month.
- **Series.dt.is_month_end** - Indicates whether the date is the last day of the month.
- **Series.dt.is_quarter_start** - Indicator for whether the date is the first day of a quarter.
- **Series.dt.is_quarter_end** - Indicator for whether the date is the last day of a quarter.
- **Series.dt.is_year_start** - Indicate whether the date is the first day of a year.
- **Series.dt.is_year_end** - Indicate whether the date is the last day of the year.

- **Series.dt.is_leap_year** - Boolean indicator if the date belongs to a leap year.
- **Series.dt.daysinmonth** - The number of days in the month.
- **Series.dt.days_in_month** - The number of days in the month.
- **Series.dt.tz** - Return the timezone.
- **Series.dt.freq** - Return the frequency object for this PeriodArray.
- **Series.dt.unit** -
- **Series.dt.isocalendar ()** - Calculate year, week, and day according to the ISO 8601 standard.
- **Series.dt.to_period (*args, **kwargs)** - Cast to PeriodArray/ PeriodIndex at a particular frequency.
- **Series.dt.to_pydatetime ()** - (DEPRECATED) Return the data as an array of datetime.datetime objects.
- **Series.dt.tz_localize (*args, **kwargs)** - Localize tz-naive Datetime Array/Index to tz-aware Datetime Array/Index.
- **Series.dt.tz_convert (*args, **kwargs)** - Convert tz-aware Datetime Array/Index from one time zone to another.
- **Series.dt.normalize (*args, **kwargs)** - Convert times to midnight.
- **Series.dt.strftime (*args, **kwargs)** - Convert to Index using specified date_format.
- **Series.dt.round (*args, **kwargs)** - Perform round operation on the data to the specified freq .
- **Series.dt.floor (*args, **kwargs)** - Perform floor operation on the data to the specified freq .
- **Series.dt.ceil (*args, **kwargs)** - Perform ceil operation on the data to the specified freq .
- **Series.dt.month_name (*args, **kwargs)** - Return the month names with specified locale.
- **Series.dt.day_name (*args, **kwargs)** - Return the day names with specified locale.
- **Series.dt.as_unit (*args, **kwargs)** -
- **Series.dt.qyear** -
- **Series.dt.start_time** - Get the Timestamp for the start of the period.
- **Series.dt.end_time** - Get the Timestamp for the end of the period.
- **Series.dt.days** - Number of days for each element.
- **Series.dt.seconds** - Number of seconds (>\= 0 and less than 1 day) for each element.
- **Series.dt.microseconds** - Number of microseconds (>\= 0 and less than 1 second) for each element.
- **Series.dt.nanoseconds** - Number of nanoseconds (>\= 0 and less than 1 microsecond) for each element.
- **Series.dt.components** - Return a Dataframe of the components of the Timedeltas.
- **Series.dt.unit** -
- **Series.dt.to_pytimedelta ()** - Return an array of native datetime.timedelta objects.
- **Series.dt.total_seconds (*args, **kwargs)** - Return total duration of each element expressed in seconds.
- **Series.dt.as_unit (*args, **kwargs)** -

- **Series.str.capitalize ()** - Convert strings in the Series/ Index to be capitalized.
- **Series.str.casefold ()** - Convert strings in the Series/Index to be casefolded.
- **Series.str.cat ([others, sep, na_rep, join])** - Concatenate strings in the Series/Index with given separator.
- **Series.str.center (width[, fillchar])** - Pad left and right side of strings in the Series/Index.
- **Series.str.contains (pat[, case, flags, na, ...])** - Test if pattern or regex is contained within a string of a Series or Index.
- **Series.str.count (pat[, flags])** - Count occurrences of pattern in each string of the Series/Index.
- **Series.str.decode (encoding[, errors])** - Decode character string in the Series/Index using indicated encoding.
- **Series.str.encode (encoding[, errors])** - Encode character string in the Series/Index using indicated encoding.
- **Series.str.endswith (pat[, na])** - Test if the end of each string element matches a pattern.
- **Series.str.extract (pat[, flags, expand])** - Extract capture groups in the regex pat as columns in a DataFrame.
- **Series.str.extractall (pat[, flags])** - Extract capture groups in the regex pat as columns in DataFrame.
- **Series.str.find (sub[, start, end])** - Return lowest indexes in each strings in the Series/Index.
- **Series.str.findall (pat[, flags])** - Find all occurrences of pattern or regular expression in the Series/Index.
- **Series.str.fullmatch (pat[, case, flags, na])** - Determine if each string entirely matches a regular expression.
- **Series.str.get (i)** - Extract element from each component at specified position or with specified key.
- **Series.str.index (sub[, start, end])** - Return lowest indexes in each string in Series/Index.
- **Series.str.join (sep)** - Join lists contained as elements in the Series/Index with passed delimiter.
- **Series.str.len ()** - Compute the length of each element in the Series/Index.
- **Series.str.ljust (width[, fillchar])** - Pad right side of strings in the Series/Index.
- **Series.str.lower ()** - Convert strings in the Series/Index to lowercase.
- **Series.str.lstrip ([to_strip])** - Remove leading characters.
- **Series.str.match (pat[, case, flags, na])** - Determine if each string starts with a match of a regular expression.
- **Series.str.normalize (form)** - Return the Unicode normal form for the strings in the Series/Index.
- **Series.str.pad (width[, side, fillchar])** - Pad strings in the Series/Index up to width.
- **Series.str.partition ([sep, expand])** - Split the string at the first occurrence of sep .
- **Series.str.removeprefix (prefix)** - Remove a prefix from an object series.

- **Series.str.removesuffix (suffix)** - Remove a suffix from an object series.
- **Series.str.repeat (repeats)** - Duplicate each string in the Series or Index.
- **Series.str.replace (pat, repl[, n, case, ...])** - Replace each occurrence of pattern/regex in the Series/Index.
- **Series.str.rfind (sub[, start, end])** - Return highest indexes in each strings in the Series/Index.
- **Series.str.rindex (sub[, start, end])** - Return highest indexes in each string in Series/Index.
- **Series.str.rjust (width[, fillchar])** - Pad left side of strings in the Series/Index.
- **Series.str.rpartition ([sep, expand])** - Split the string at the last occurrence of sep .
- **Series.str.rstrip ([to_strip])** - Remove trailing characters.
- **Series.str.slice ([start, stop, step])** - Slice substrings from each element in the Series or Index.
- **Series.str.slice_replace ([start, stop, repl])** - Replace a positional slice of a string with another value.
- **Series.str.split ([pat, n, expand, regex])** - Split strings around given separator/delimiter.
- **Series.str.rsplit ([pat, n, expand])** - Split strings around given separator/delimiter.
- **Series.str.startswith (pat[, na])** - Test if the start of each string element matches a pattern.
- **Series.str.strip ([to_strip])** - Remove leading and trailing characters.
- **Series.str.swapcase ()** - Convert strings in the Series/Index to be swapcased.
- **Series.str.title ()** - Convert strings in the Series/Index to titlecase.
- **Series.str.translate (table)** - Map all characters in the string through the given mapping table.
- **Series.str.upper ()** - Convert strings in the Series/Index to uppercase.
- **Series.str.wrap (width, **kwargs)** - Wrap strings in Series/ Index at specified line width.
- **Series.str.zfill (width)** - Pad strings in the Series/Index by prepending '0' characters.
- **Series.str.isalnum ()** - Check whether all characters in each string are alphanumeric.
- **Series.str.isalpha ()** - Check whether all characters in each string are alphabetic.
- **Series.str.isdigit ()** - Check whether all characters in each string are digits.
- **Series.str.isspace ()** - Check whether all characters in each string are whitespace.
- **Series.str.islower ()** - Check whether all characters in each string are lowercase.
- **Series.str.isupper ()** - Check whether all characters in each string are uppercase.
- **Series.str.istitle ()** - Check whether all characters in each string are titlecase.

- **Series.str.isnumeric ()** - Check whether all characters in each string are numeric.
- **Series.str.isdecimal ()** - Check whether all characters in each string are decimal.
- **Series.str.get_dummies ([sep])** - Return DataFrame of dummy/ indicator variables for Series.
- **Series.cat.categories** - The categories of this categorical.
- **Series.cat.ordered** - Whether the categories have an ordered relationship.
- **Series.cat.codes** - Return Series of codes as well as the index.
- **Series.cat.rename_categories (*args, **kwargs)** - Rename categories.
- **Series.cat.reorder_categories (*args, **kwargs)** - Reorder categories as specified in new_categories.
- **Series.cat.add_categories (*args, **kwargs)** - Add new categories.
- **Series.cat.remove_categories (*args, **kwargs)** - Remove the specified categories.
- **Series.cat.remove_unused_categories (*args, ...)** - Remove categories which are not used.
- **Series.cat.set_categories (*args, **kwargs)** - Set the categories to the specified new categories.
- **Series.cat.as_ordered (*args, **kwargs)** - Set the Categorical to be ordered.
- **Series.cat.as_unordered (*args, **kwargs)** - Set the Categorical to be unordered.
- **Series.sparse.npoints** - The number of non- fill_value points.
- **Series.sparse.density** - The percent of non- fill_value points, as decimal.
- **Series.sparse.fill_value** - Elements in data that are fill_value are not stored.
- **Series.sparse.sp_values** - An ndarray containing the non- fill_value values.
- **Series.sparse.from_coo (A[, dense_index])** - Create a Series with sparse values from a scipy.sparse.coo_matrix.
- **Series.sparse.to_coo ([row_levels, ...])** - Create a scipy.sparse.coo_matrix from a Series with MultiIndex.
- **Series.list.flatten ()** - Flatten list values.
- **Series.list.len ()** - Return the length of each list in the Series.
- **Series.list.__getitem__ (key)** - Index or slice lists in the Series.
- **Series.struct.dtypes** - Return the dtype object of each child field of the struct.
- **Series.struct.field (name_or_index)** - Extract a child field of a struct as a Series.
- **Series.struct.explode ()** - Extract all child fields of a struct as a DataFrame.
- **Flags (obj, *, allows_duplicate_labels)** - Flags that apply to pandas objects.

- **Series.attrs** - Dictionary of global attributes of this dataset.

Plotting

- **Series.plot ([kind, ax, figsize, ...])** - Series plotting accessor and method

- **Series.plot.area ([x, y, stacked])** - Draw a stacked area plot.
- **Series.plot.bar ([x, y])** - Vertical bar plot.
- **Series.plot.barh ([x, y])** - Make a horizontal bar plot.
- **Series.plot.box ([by])** - Make a box plot of the DataFrame columns.
- **Series.plot.density ([bw_method, ind])** - Generate Kernel Density Estimate plot using Gaussian kernels.
- **Series.plot.hist ([by, bins])** - Draw one histogram of the DataFrame's columns.
- **Series.plot.kde ([bw_method, ind])** - Generate Kernel Density Estimate plot using Gaussian kernels.
- **Series.plot.line ([x, y])** - Plot Series or DataFrame as lines.
- **Series.plot.pie (**kwargs)** - Generate a pie plot.

- **Series.hist ([by, ax, grid, xlabelsize, ...])** - Draw histogram of the input series using matplotlib.

Serialization / IO / conversion

- **Series.to_pickle (path, *[, compression, ...])** - Pickle (serialize) object to file.

- **Series.to_csv ([path_or_buf, sep, na_rep, ...])** - Write object to a comma-separated values (csv) file.
- **Series.to_dict (*[, into])** - Convert Series to {label -> value} dict or dict-like object.
- **Series.to_excel (excel_writer, *[, ...])** - Write object to an Excel sheet.
- **Series.to_frame ([name])** - Convert Series to DataFrame.
- **Series.to_xarray ()** - Return an xarray object from the pandas object.
- **Series.to_hdf (path_or_buf, *, key[, mode, ...])** - Write the contained data to an HDF5 file using HDFStore.
- **Series.to_sql (name, con, *[, schema, ...])** - Write records stored in a DataFrame to a SQL database.
- **Series.to_json ([path_or_buf, orient, ...])** - Convert the object to a JSON string.
- **Series.to_string ([buf, na_rep, ...])** - Render a string representation of the Series.
- **Series.to_clipboard (*[, excel, sep])** - Copy object to the system clipboard.

- **Series.to_latex ([buf, columns, header, ...])** - Render object to a LaTeX tabular, longtable, or nested table.
- **Series.to_markdown ([buf, mode, index, ...])** - Print Series in Markdown-friendly format.

DataFrame

Constructor

- **DataFrame ([data, index, columns, dtype, copy])** - Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Attributes and underlying data

- **DataFrame.index** - The index (row labels) of the DataFrame.

- **DataFrame.columns** - The column labels of the DataFrame.
- **DataFrame.dtypes** - Return the dtypes in the DataFrame.
- **DataFrame.info ([verbose, buf, max_cols, ...])** - Print a concise summary of a DataFrame.
- **DataFrame.select_dtypes ([include, exclude])** - Return a subset of the DataFrame's columns based on the column dtypes.
- **DataFrame.values** - Return a Numpy representation of the DataFrame.
- **DataFrame.axes** - Return a list representing the axes of the DataFrame.
- **DataFrame.ndim** - Return an int representing the number of axes / array dimensions.
- **DataFrame.size** - Return an int representing the number of elements in this object.
- **DataFrame.shape** - Return a tuple representing the dimensionality of the DataFrame.
- **DataFrame.memory_usage ([index, deep])** - Return the memory usage of each column in bytes.
- **DataFrame.empty** - Indicator whether Series/DataFrame is empty.

- **DataFrame.set_flags (*[, copy, ...])** - Return a new object with updated flags.

Conversion

- **DataFrame.astype (dtype[, copy, errors])** - Cast a pandas object to a specified dtype dtype .

- **DataFrame.convert_dtypes ([infer_objects, ...])** - Convert columns to the best possible dtypes using dtypes supporting pd.NA .

- **DataFrame.infer_objects ([copy])** - Attempt to infer better dtypes for object columns.
- **DataFrame.copy ([deep])** - Make a copy of this object's indices and data.
- **DataFrame.bool ()** - (DEPRECATED) Return the bool of a single element Series or DataFrame.

- **DataFrame.to_numpy ([dtype, copy, na_value])** - Convert the DataFrame to a NumPy array.

Indexing, iteration

- **DataFrame.head ([n])** - Return the first n rows.

- **DataFrame.at** - Access a single value for a row/column label pair.
- **DataFrame.iat** - Access a single value for a row/column pair by integer position.
- **DataFrame.loc** - Access a group of rows and columns by label(s) or a boolean array.
- **DataFrame.iloc** - (DEPRECATED) Purely integer-location based indexing for selection by position.
- **DataFrame.insert (loc, column, value[, ...])** - Insert column into DataFrame at specified location.
- **DataFrame.__iter__ ()** - Iterate over info axis.
- **DataFrame.items ()** - Iterate over (column name, Series) pairs.
- **DataFrame.keys ()** - Get the 'info axis' (see Indexing for more).
- **DataFrame.iterrows ()** - Iterate over DataFrame rows as (index, Series) pairs.
- **DataFrame.itertuples ([index, name])** - Iterate over DataFrame rows as namedtuples.
- **DataFrame.pop (item)** - Return item and drop from frame.
- **DataFrame.tail ([n])** - Return the last n rows.
- **DataFrame.xs (key[, axis, level, drop_level])** - Return cross-section from the Series/DataFrame.
- **DataFrame.get (key[, default])** - Get item from object for given key (ex: DataFrame column).
- **DataFrame.isin (values)** - Whether each element in the DataFrame is contained in values.
- **DataFrame.where (cond[, other, inplace, ...])** - Replace values where the condition is False.
- **DataFrame.mask (cond[, other, inplace, axis, ...])** - Replace values where the condition is True.

- **DataFrame.query (expr, *[, inplace])** - Query the columns of a DataFrame with a boolean expression.

Binary operator functions

- **DataFrame.__add__ (other)** - Get Addition of DataFrame and other, column-wise.

- **DataFrame.add (other[, axis, level, fill_value])** - Get Addition of dataframe and other, element-wise (binary operator add).
- **DataFrame.sub (other[, axis, level, fill_value])** - Get Subtraction of dataframe and other, element-wise (binary operator sub).
- **DataFrame.mul (other[, axis, level, fill_value])** - Get Multiplication of dataframe and other, element-wise (binary operator mul).
- **DataFrame.div (other[, axis, level, fill_value])** - Get Floating division of dataframe and other, element-wise (binary operator truediv).
- **DataFrame.truediv (other[, axis, level, ...])** - Get Floating division of dataframe and other, element-wise (binary operator truediv).
- **DataFrame.floordiv (other[, axis, level, ...])** - Get Integer division of dataframe and other, element-wise (binary operator floordiv).
- **DataFrame.mod (other[, axis, level, fill_value])** - Get Modulo of dataframe and other, element-wise (binary operator mod).
- **DataFrame.pow (other[, axis, level, fill_value])** - Get Exponential power of dataframe and other, element-wise (binary operator pow).
- **DataFrame.dot (other)** - Compute the matrix multiplication between the DataFrame and other.
- **DataFrame.radd (other[, axis, level, fill_value])** - Get Addition of dataframe and other, element-wise (binary operator radd).
- **DataFrame.rsub (other[, axis, level, fill_value])** - Get Subtraction of dataframe and other, element-wise (binary operator rsub).
- **DataFrame.rmul (other[, axis, level, fill_value])** - Get Multiplication of dataframe and other, element-wise (binary operator rmul).
- **DataFrame.rdiv (other[, axis, level, fill_value])** - Get Floating division of dataframe and other, element-wise (binary operator rtruediv).
- **DataFrame.rtruediv (other[, axis, level, ...])** - Get Floating division of dataframe and other, element-wise (binary operator rtruediv).
- **DataFrame.rfloordiv (other[, axis, level, ...])** - Get Integer division of dataframe and other, element-wise (binary operator rfloordiv).
- **DataFrame.rmod (other[, axis, level, fill_value])** - Get Modulo of dataframe and other, element-wise (binary operator rmod).

- **DataFrame.rpow (other[, axis, level, fill_value])** - Get Exponential power of dataframe and other, element-wise (binary operator rpow).
- **DataFrame.lt (other[, axis, level])** - Get Less than of dataframe and other, element-wise (binary operator lt).
- **DataFrame.gt (other[, axis, level])** - Get Greater than of dataframe and other, element-wise (binary operator gt).
- **DataFrame.le (other[, axis, level])** - Get Less than or equal to of dataframe and other, element-wise (binary operator le).
- **DataFrame.ge (other[, axis, level])** - Get Greater than or equal to of dataframe and other, element-wise (binary operator ge).
- **DataFrame.ne (other[, axis, level])** - Get Not equal to of dataframe and other, element-wise (binary operator ne).
- **DataFrame.eq (other[, axis, level])** - Get Equal to of dataframe and other, element-wise (binary operator eq).
- **DataFrame.combine (other, func[, fill_value, ...])** - Perform column-wise combine with another DataFrame.

- **DataFrame.combine_first (other)** - Update null elements with value in the same location in other .

Function application, GroupBy & window

- **DataFrame.apply (func[, axis, raw, ...])** - Apply a function along an axis of the DataFrame.

- **DataFrame.map (func[, na_action])** - Apply a function to a Dataframe elementwise.
- **DataFrame.applymap (func[, na_action])** - (DEPRECATED) Apply a function to a Dataframe elementwise.
- **DataFrame.pipe (func, *args, **kwargs)** - Apply chainable functions that expect Series or DataFrames.
- **DataFrame.agg ([func, axis])** - Aggregate using one or more operations over the specified axis.
- **DataFrame.aggregate ([func, axis])** - Aggregate using one or more operations over the specified axis.
- **DataFrame.transform (func[, axis])** - Call func on self producing a DataFrame with the same axis shape as self.
- **DataFrame.groupby ([by, axis, level, ...])** - Group DataFrame using a mapper or by a Series of columns.
- **DataFrame.rolling (window[, min_periods, ...])** - Provide rolling window calculations.
- **DataFrame.expanding ([min_periods, axis, method])** - Provide expanding window calculations.

- **DataFrame.ewm ([com, span, halflife, alpha, ...])** - Provide exponentially weighted (EW) calculations.

Computations / descriptive stats

- **DataFrame.abs ()** - Return a Series/DataFrame with absolute numeric value of each element.

- **DataFrame.all ([axis, bool_only, skipna])** - Return whether all elements are True, potentially over an axis.
- **DataFrame.any (*[, axis, bool_only, skipna])** - Return whether any element is True, potentially over an axis.
- **DataFrame.clip ([lower, upper, axis, inplace])** - Trim values at input threshold(s).
- **DataFrame.corr ([method, min_periods, ...])** - Compute pairwise correlation of columns, excluding NA/null values.
- **DataFrame.corrwith (other[, axis, drop, ...])** - Compute pairwise correlation.
- **DataFrame.count ([axis, numeric_only])** - Count non-NA cells for each column or row.
- **DataFrame.cov ([min_periods, ddof, numeric_only])** - Compute pairwise covariance of columns, excluding NA/null values.
- **DataFrame.cummax ([axis, skipna])** - Return cumulative maximum over a DataFrame or Series axis.
- **DataFrame.cummin ([axis, skipna])** - Return cumulative minimum over a DataFrame or Series axis.
- **DataFrame.cumprod ([axis, skipna])** - Return cumulative product over a DataFrame or Series axis.
- **DataFrame.cumsum ([axis, skipna])** - Return cumulative sum over a DataFrame or Series axis.
- **DataFrame.describe ([percentiles, include, ...])** - Generate descriptive statistics.
- **DataFrame.diff ([periods, axis])** - First discrete difference of element.
- **DataFrame.eval (expr, *[, inplace])** - Evaluate a string describing operations on DataFrame columns.
- **DataFrame.kurt ([axis, skipna, numeric_only])** - Return unbiased kurtosis over requested axis.
- **DataFrame.kurtosis ([axis, skipna, numeric_only])** - Return unbiased kurtosis over requested axis.
- **DataFrame.max ([axis, skipna, numeric_only])** - Return the maximum of the values over the requested axis.
- **DataFrame.mean ([axis, skipna, numeric_only])** - Return the mean of the values over the requested axis.
- **DataFrame.median ([axis, skipna, numeric_only])** - Return the median of the values over the requested axis.
- **DataFrame.min ([axis, skipna, numeric_only])** - Return the minimum of the values over the requested axis.
- **DataFrame.mode ([axis, numeric_only, dropna])** - Get the mode(s) of each element along the selected axis.
- **DataFrame.pct_change ([periods, fill_method, ...])** - Fractional change between the current and a prior element.
- **DataFrame.prod ([axis, skipna, numeric_only, ...])** - Return the product of the values over the requested axis.
- **DataFrame.product ([axis, skipna, ...])** - Return the product of the values over the requested axis.

- **DataFrame.quantile ([q, axis, numeric_only, ...])** - Return values at the given quantile over requested axis.
- **DataFrame.rank ([axis, method, numeric_only, ...])** - Compute numerical data ranks (1 through n) along axis.
- **DataFrame.round ([decimals])** - Round a DataFrame to a variable number of decimal places.
- **DataFrame.sem ([axis, skipna, ddof, numeric_only])** - Return unbiased standard error of the mean over requested axis.
- **DataFrame.skew ([axis, skipna, numeric_only])** - Return unbiased skew over requested axis.
- **DataFrame.sum ([axis, skipna, numeric_only, ...])** - Return the sum of the values over the requested axis.
- **DataFrame.std ([axis, skipna, ddof, numeric_only])** - Return sample standard deviation over requested axis.
- **DataFrame.var ([axis, skipna, ddof, numeric_only])** - Return unbiased variance over requested axis.
- **DataFrame.nunique ([axis, dropna])** - Count number of distinct elements in specified axis.

- **DataFrame.value_counts ([subset, normalize, ...])** - Return a Series containing the frequency of each distinct row in the Dataframe.

Reindexing / selection / label manipulation

- **DataFrame.add_prefix (prefix[, axis])** - Prefix labels with string prefix .

- **DataFrame.add_suffix (suffix[, axis])** - Suffix labels with string suffix .
- **DataFrame.align (other[, join, axis, level, ...])** - Align two objects on their axes with the specified join method.
- **DataFrame.at_time (time[, asof, axis])** - Select values at particular time of day (e.g., 9:30AM).
- **DataFrame.between_time (start_time, end_time)** - Select values between particular times of the day (e.g., 9:00-9:30 AM).
- **DataFrame.drop ([labels, axis, index, ...])** - Drop specified labels from rows or columns.
- **DataFrame.drop_duplicates ([subset, keep, ...])** - Return DataFrame with duplicate rows removed.
- **DataFrame.duplicated ([subset, keep])** - Return boolean Series denoting duplicate rows.
- **DataFrame.equals (other)** - Test whether two objects contain the same elements.
- **DataFrame.filter ([items, like, regex, axis])** - Subset the dataframe rows or columns according to the specified index labels.
- **DataFrame.first (offset)** - (DEPRECATED) Select initial periods of time series data based on a date offset.
- **DataFrame.head ([n])** - Return the first n rows.

- **DataFrame.idxmax ([axis, skipna, numeric_only])** - Return index of first occurrence of maximum over requested axis.
- **DataFrame.idxmin ([axis, skipna, numeric_only])** - Return index of first occurrence of minimum over requested axis.
- **DataFrame.last (offset)** - (DEPRECATED) Select final periods of time series data based on a date offset.
- **DataFrame.reindex ([labels, index, columns, ...])** - Conform DataFrame to new index with optional filling logic.
- **DataFrame.reindex_like (other[, method, ...])** - Return an object with matching indices as other object.
- **DataFrame.rename ([mapper, index, columns, ...])** - Rename columns or index labels.
- **DataFrame.rename_axis ([mapper, index, ...])** - Set the name of the axis for the index or columns.
- **DataFrame.reset_index ([level, drop, ...])** - Reset the index, or a level of it.
- **DataFrame.sample ([n, frac, replace, ...])** - Return a random sample of items from an axis of object.
- **DataFrame.set_axis (labels, *[, axis, copy])** - Assign desired index to given axis.
- **DataFrame.set_index (keys, *[, drop, append, ...])** - Set the DataFrame index using existing columns.
- **DataFrame.tail ([n])** - Return the last n rows.
- **DataFrame.take (indices[, axis])** - Return the elements in the given positional indices along an axis.

- **DataFrame.truncate ([before, after, axis, copy])** - Truncate a Series or DataFrame before and after some index value.

Missing data handling

- **DataFrame.backfill (*[, axis, inplace, ...])** - (DEPRECATED) Fill NA/NaN values by using the next valid observation to fill the gap.

- **DataFrame.bfill (*[, axis, inplace, limit, ...])** - Fill NA/NaN values by using the next valid observation to fill the gap.
- **DataFrame.dropna (*[, axis, how, thresh, ...])** - Remove missing values.
- **DataFrame.ffill (*[, axis, inplace, limit, ...])** - Fill NA/NaN values by propagating the last valid observation to next valid.
- **DataFrame.fillna ([value, method, axis, ...])** - Fill NA/NaN values using the specified method.
- **DataFrame.interpolate ([method, axis, limit, ...])** - Fill NaN values using an interpolation method.
- **DataFrame.isna ()** - Detect missing values.
- **DataFrame.isnull ()** - DataFrame.isnull is an alias for DataFrame.isna.
- **DataFrame.notna ()** - Detect existing (non-missing) values.
- **DataFrame.notnull ()** - DataFrame.notnull is an alias for DataFrame.notna.

- **DataFrame.pad (*[, axis, inplace, limit, ...])** - (DEPRECATED) Fill NA/NaN values by propagating the last valid observation to next valid.
- **DataFrame.replace ([to_replace, value, ...])** - Replace values given in to_replace with value .

Reshaping, sorting, transposing

- **DataFrame.droplevel (level[, axis])** - Return Series/DataFrame with requested index / column level(s) removed.
- **DataFrame.pivot (*, columns[, index, values])** - Return reshaped DataFrame organized by given index / column values.
- **DataFrame.pivot_table ([values, index, ...])** - Create a spreadsheet-style pivot table as a DataFrame.
- **DataFrame.reorder_levels (order[, axis])** - Rearrange index levels using input order.
- **DataFrame.sort_values (by, *[, axis, ...])** - Sort by the values along either axis.
- **DataFrame.sort_index (*[, axis, level, ...])** - Sort object by labels (along an axis).
- **DataFrame.nlargest (n, columns[, keep])** - Return the first n rows ordered by columns in descending order.
- **DataFrame.nsmallest (n, columns[, keep])** - Return the first n rows ordered by columns in ascending order.
- **DataFrame.swaplevel ([i, j, axis])** - Swap levels i and j in a MultiIndex .
- **DataFrame.stack ([level, dropna, sort, ...])** - Stack the prescribed level(s) from columns to index.
- **DataFrame.unstack ([level, fill_value, sort])** - Pivot a level of the (necessarily hierarchical) index labels.
- **DataFrame.swapaxes (axis1, axis2[, copy])** - (DEPRECATED) Interchange axes and swap values axes appropriately.
- **DataFrame.melt ([id_vars, value_vars, ...])** - Unpivot a DataFrame from wide to long format, optionally leaving identifiers set.
- **DataFrame.explode (column[, ignore_index])** - Transform each element of a list-like to a row, replicating index values.
- **DataFrame.squeeze ([axis])** - Squeeze 1 dimensional axis objects into scalars.
- **DataFrame.to_xarray ()** - Return an xarray object from the pandas object.
- **DataFrame.T** - The transpose of the DataFrame.
- **DataFrame.transpose (*args[, copy])** - Transpose index and columns.

Combining / comparing / joining / merging

- **DataFrame.assign (**kwargs)** - Assign new columns to a DataFrame.

- **DataFrame.compare (other[, align_axis, ...])** - Compare to another DataFrame and show the differences.
- **DataFrame.join (other[, on, how, lsuffix, ...])** - Join columns of another DataFrame.
- **DataFrame.merge (right[, how, on, left_on, ...])** - Merge DataFrame or named Series objects with a database-style join.

- **DataFrame.update (other[, join, overwrite, ...])** - Modify in place using non-NA values from another DataFrame.

Time Series-related

- **DataFrame.asfreq (freq[, method, how, ...])** - Convert time series to specified frequency.

- **DataFrame.asof (where[, subset])** - Return the last row(s) without any NaNs before where .
- **DataFrame.shift ([periods, freq, axis, ...])** - Shift index by desired number of periods with an optional time freq .
- **DataFrame.first_valid_index ()** - Return index for first non-NA value or None, if no non-NA value is found.
- **DataFrame.last_valid_index ()** - Return index for last non-NA value or None, if no non-NA value is found.
- **DataFrame.resample (rule[, axis, closed, ...])** - Resample time-series data.
- **DataFrame.to_period ([freq, axis, copy])** - Convert DataFrame from DatetimeIndex to PeriodIndex.
- **DataFrame.to_timestamp ([freq, how, axis, copy])** - Cast to DatetimeIndex of timestamps, at beginning of period.
- **DataFrame.tz_convert (tz[, axis, level, copy])** - Convert tz-aware axis to target time zone.

- **DataFrame.tz_localize (tz[, axis, level, ...])** - Localize tz-naive index of a Series or DataFrame to target time zone.

Flags

- **Flags (obj, *, allows_duplicate_labels)** - Flags that apply to pandas objects.

Metadata

- **DataFrame.attrs** - Dictionary of global attributes of this dataset.

Plotting

- **DataFrame.plot ([x, y, kind, ax, ...])** - DataFrame plotting accessor and method

- **DataFrame.plot.area ([x, y, stacked])** - Draw a stacked area plot.
- **DataFrame.plot.bar ([x, y])** - Vertical bar plot.
- **DataFrame.plot.barh ([x, y])** - Make a horizontal bar plot.
- **DataFrame.plot.box ([by])** - Make a box plot of the DataFrame columns.
- **DataFrame.plot.density ([bw_method, ind])** - Generate Kernel Density Estimate plot using Gaussian kernels.
- **DataFrame.plot.hexbin (x, y[, C, ...])** - Generate a hexagonal binning plot.
- **DataFrame.plot.hist ([by, bins])** - Draw one histogram of the DataFrame's columns.
- **DataFrame.plot.kde ([bw_method, ind])** - Generate Kernel Density Estimate plot using Gaussian kernels.
- **DataFrame.plot.line ([x, y])** - Plot Series or DataFrame as lines.
- **DataFrame.plot.pie (**kwargs)** - Generate a pie plot.
- **DataFrame.plot.scatter (x, y[, s, c])** - Create a scatter plot with varying marker point size and color.
- **DataFrame.boxplot ([column, by, ax, ...])** - Make a box plot from DataFrame columns.

- **DataFrame.hist ([column, by, grid, ...])** - Make a histogram of the DataFrame's columns.

Sparse accessor

- **DataFrame.sparse.density** - Ratio of non-sparse points to total (dense) data points.

- **DataFrame.sparse.from_spmatrix (data[, ...])** - Create a new DataFrame from a scipy sparse matrix.
- **DataFrame.sparse.to_coo ()** - Return the contents of the frame as a sparse SciPy COO matrix.

- **DataFrame.sparse.to_dense ()** - Convert a DataFrame with sparse values to dense.

Serialization / IO / conversion

- **DataFrame.from_dict (data[, orient, dtype, ...])** - Construct DataFrame from dict of array-like or dicts.

- **DataFrame.from_records (data[, index, ...])** - Convert structured or record ndarray to DataFrame.
- **DataFrame.to_orc ([path, engine, index, ...])** - Write a DataFrame to the ORC format.

- **DataFrame.to_parquet ([path, engine, ...])** - Write a DataFrame to the binary parquet format.
- **DataFrame.to_pickle (path, *[, compression, ...])** - Pickle (serialize) object to file.
- **DataFrame.to_csv ([path_or_buf, sep, na_rep, ...])** - Write object to a comma-separated values (csv) file.
- **DataFrame.to_hdf (path_or_buf, *, key[, ...])** - Write the contained data to an HDF5 file using HDFStore.
- **DataFrame.to_sql (name, con, *[, schema, ...])** - Write records stored in a DataFrame to a SQL database.
- **DataFrame.to_dict ([orient, into, index])** - Convert the DataFrame to a dictionary.
- **DataFrame.to_excel (excel_writer, *[, ...])** - Write object to an Excel sheet.
- **DataFrame.to_json ([path_or_buf, orient, ...])** - Convert the object to a JSON string.
- **DataFrame.to_html ([buf, columns, col_space, ...])** - Render a DataFrame as an HTML table.
- **DataFrame.to_feather (path, **kwargs)** - Write a DataFrame to the binary Feather format.
- **DataFrame.to_latex ([buf, columns, header, ...])** - Render object to a LaTeX tabular, longtable, or nested table.
- **DataFrame.to_stata (path, *[, convert_dates, ...])** - Export DataFrame object to Stata dta format.
- **DataFrame.to_gbq (destination_table, *[, ...])** - (DEPRECATED) Write a DataFrame to a Google BigQuery table.
- **DataFrame.to_records ([index, column_dtypes, ...])** - Convert DataFrame to a NumPy record array.
- **DataFrame.to_string ([buf, columns, ...])** - Render a DataFrame to a console-friendly tabular output.
- **DataFrame.to_clipboard (*[, excel, sep])** - Copy object to the system clipboard.
- **DataFrame.to_markdown ([buf, mode, index, ...])** - Print DataFrame in Markdown-friendly format.
- **DataFrame.style** - Returns a Styler object.
- **DataFrame.__dataframe__ ([nan_as_null, ...])** - Return the dataframe interchange object implementing the interchange protocol.

pandas arrays, scalars, and data types

Objects

- **array (data[, dtype, copy])** - Create an array.
- **arrays.ArrowExtensionArray (values)** - Pandas ExtensionArray backed by a PyArrow ChunkedArray.
- **ArrowDtype (pyarrow_dtype)** - An ExtensionDtype for PyArrow data types.

- **Timestamp ([ts_input, year, month, day, ...])** - Pandas replacement for python datetime.datetime object.
- **Timestamp.asm8** - Return numpy datetime64 format in nanoseconds.
- **Timestamp.day** -
- **Timestamp.dayofweek** - Return day of the week.
- **Timestamp.day_of_week** - Return day of the week.
- **Timestamp.dayofyear** - Return the day of the year.
- **Timestamp.day_of_year** - Return the day of the year.
- **Timestamp.days_in_month** - Return the number of days in the month.
- **Timestamp.daysinmonth** - Return the number of days in the month.
- **Timestamp.fold** -
- **Timestamp.hour** -
- **Timestamp.is_leap_year** - Return True if year is a leap year.
- **Timestamp.is_month_end** - Check if the date is the last day of the month.
- **Timestamp.is_month_start** - Check if the date is the first day of the month.
- **Timestamp.is_quarter_end** - Check if date is last day of the quarter.
- **Timestamp.is_quarter_start** - Check if the date is the first day of the quarter.
- **Timestamp.is_year_end** - Return True if date is last day of the year.
- **Timestamp.is_year_start** - Return True if date is first day of the year.
- **Timestamp.max** -
- **Timestamp.microsecond** -
- **Timestamp.min** -
- **Timestamp.minute** -
- **Timestamp.month** -
- **Timestamp.nanosecond** -
- **Timestamp.quarter** - Return the quarter of the year.
- **Timestamp.resolution** -
- **Timestamp.second** -
- **Timestamp.tz** - Alias for tzinfo.
- **Timestamp.tzinfo** -
- **Timestamp.unit** - The abbreviation associated with self._creso.
- **Timestamp.value** -
- **Timestamp.week** - Return the week number of the year.
- **Timestamp.weekofyear** - Return the week number of the year.
- **Timestamp.year** -
- **Timestamp.as_unit (unit[, round_ok])** - Convert the underlying int64 representaton to the given unit.
- **Timestamp.astimezone (tz)** - Convert timezone-aware Timestamp to another time zone.
- **Timestamp.ceil (freq[, ambiguous, nonexistent])** - Return a new Timestamp ceiled to this resolution.
- **Timestamp.combine (date, time)** - Combine date, time into datetime with same date and time fields.

- **Timestamp.ctime ()** - Return ctime() style string.
- **Timestamp.date ()** - Return date object with same year, month and day.
- **Timestamp.day_name ([locale])** - Return the day name of the Timestamp with specified locale.
- **Timestamp.dst ()** - Return the daylight saving time (DST) adjustment.
- **Timestamp.floor (freq[, ambiguous, nonexistent])** - Return a new Timestamp floored to this resolution.
- **Timestamp.fromordinal (ordinal[, tz])** - Construct a timestamp from a a proleptic Gregorian ordinal.
- **Timestamp.fromtimestamp (ts)** - Transform timestamp[, tz] to tz's local time from POSIX timestamp.
- **Timestamp.isocalendar ()** - Return a named tuple containing ISO year, week number, and weekday.
- **Timestamp.isoformat ([sep, timespec])** - Return the time formatted according to ISO 8601.
- **Timestamp.isoweekday ()** - Return the day of the week represented by the date.
- **Timestamp.month_name ([locale])** - Return the month name of the Timestamp with specified locale.
- **Timestamp.normalize ()** - Normalize Timestamp to midnight, preserving tz information.
- **Timestamp.now ([tz])** - Return new Timestamp object representing current time local to tz.
- **Timestamp.replace ([year, month, day, hour, ...])** - Implements datetime.replace, handles nanoseconds.
- **Timestamp.round (freq[, ambiguous, nonexistent])** - Round the Timestamp to the specified resolution.
- **Timestamp.strftime (format)** - Return a formatted string of the Timestamp.
- **Timestamp.strptime (string, format)** - Function is not implemented.
- **Timestamp.time ()** - Return time object with same time but with tzinfo\=None.
- **Timestamp.timestamp ()** - Return POSIX timestamp as float.
- **Timestamp.timetuple ()** - Return time tuple, compatible with time.localtime().
- **Timestamp.timetz ()** - Return time object with same time and tzinfo.
- **Timestamp.to_datetime64 ()** - Return a numpy.datetime64 object with same precision.
- **Timestamp.to_numpy ([dtype, copy])** - Convert the Timestamp to a NumPy datetime64.
- **Timestamp.to_julian_date ()** - Convert TimeStamp to a Julian Date.
- **Timestamp.to_period ([freq])** - Return an period of which this timestamp is an observation.
- **Timestamp.to_pydatetime ([warn])** - Convert a Timestamp object to a native Python datetime object.
- **Timestamp.today ([tz])** - Return the current time in the local timezone.
- **Timestamp.toordinal ()** - Return proleptic Gregorian ordinal.

- **Timestamp.tz_convert (tz)** - Convert timezone-aware Timestamp to another time zone.
- **Timestamp.tz_localize (tz[, ambiguous, ...])** - Localize the Timestamp to a timezone.
- **Timestamp.tzname ()** - Return time zone name.
- **Timestamp.utcfromtimestamp (ts)** - Construct a timezone-aware UTC datetime from a POSIX timestamp.
- **Timestamp.utcnow ()** - Return a new Timestamp representing UTC day and time.
- **Timestamp.utcoffset ()** - Return utc offset.
- **Timestamp.utctimetuple ()** - Return UTC time tuple, compatible with time.localtime().
- **Timestamp.weekday ()** - Return the day of the week represented by the date.
- **arrays.DatetimeArray (values[, dtype, freq, copy])** - Pandas ExtensionArray for tz-naive or tz-aware datetime data.
- **DatetimeTZDtype ([unit, tz])** - An ExtensionDtype for timezone-aware datetime data.
- **Timedelta ([value, unit])** - Represents a duration, the difference between two dates or times.
- **Timedelta.asm8** - Return a numpy timedelta64 array scalar view.
- **Timedelta.components** - Return a components namedtuple-like.
- **Timedelta.days** - Returns the days of the timedelta.
- **Timedelta.max** -
- **Timedelta.microseconds** -
- **Timedelta.min** -
- **Timedelta.nanoseconds** - Return the number of nanoseconds (n), where 0 <\= n < 1 microsecond.
- **Timedelta.resolution** -
- **Timedelta.seconds** - Return the total hours, minutes, and seconds of the timedelta as seconds.
- **Timedelta.unit** -
- **Timedelta.value** -
- **Timedelta.view (dtype)** - Array view compatibility.
- **Timedelta.as_unit (unit[, round_ok])** - Convert the underlying int64 representation to the given unit.
- **Timedelta.ceil (freq)** - Return a new Timedelta ceiled to this resolution.
- **Timedelta.floor (freq)** - Return a new Timedelta floored to this resolution.
- **Timedelta.isoformat ()** - Format the Timedelta as ISO 8601 Duration.
- **Timedelta.round (freq)** - Round the Timedelta to the specified resolution.
- **Timedelta.to_pytimedelta ()** - Convert a pandas Timedelta object into a python datetime.timedelta object.
- **Timedelta.to_timedelta64 ()** - Return a numpy.timedelta64 object with 'ns' precision.
- **Timedelta.to_numpy ([dtype, copy])** - Convert the Timedelta to a NumPy timedelta64.
- **Timedelta.total_seconds ()** - Total seconds in the duration.

- **arrays.TimedeltaArray (values[, dtype, freq, ...])** - Pandas ExtensionArray for timedelta data.
- **Period ([value, freq, ordinal, year, month, ...])** - Represents a period of time.
- **Period.day** - Get day of the month that a Period falls on.
- **Period.dayofweek** - Day of the week the period lies in, with Monday\=0 and Sunday\=6.
- **Period.day_of_week** - Day of the week the period lies in, with Monday\=0 and Sunday\=6.
- **Period.dayofyear** - Return the day of the year.
- **Period.day_of_year** - Return the day of the year.
- **Period.days_in_month** - Get the total number of days in the month that this period falls on.
- **Period.daysinmonth** - Get the total number of days of the month that this period falls on.
- **Period.end_time** - Get the Timestamp for the end of the period.
- **Period.freq** -
- **Period.freqstr** - Return a string representation of the frequency.
- **Period.hour** - Get the hour of the day component of the Period.
- **Period.is_leap_year** - Return True if the period's year is in a leap year.
- **Period.minute** - Get minute of the hour component of the Period.
- **Period.month** - Return the month this Period falls on.
- **Period.ordinal** -
- **Period.quarter** - Return the quarter this Period falls on.
- **Period.qyear** - Fiscal year the Period lies in according to its starting-quarter.
- **Period.second** - Get the second component of the Period.
- **Period.start_time** - Get the Timestamp for the start of the period.
- **Period.week** - Get the week of the year on the given Period.
- **Period.weekday** - Day of the week the period lies in, with Monday\=0 and Sunday\=6.
- **Period.weekofyear** - Get the week of the year on the given Period.
- **Period.year** - Return the year this Period falls on.
- **Period.asfreq (freq[, how])** - Convert Period to desired frequency, at the start or end of the interval.
- **Period.now (freq)** - Return the period of now's date.
- **Period.strftime (fmt)** - Returns a formatted string representation of the Period .
- **Period.to_timestamp ([freq, how])** - Return the Timestamp representation of the Period.
- **arrays.PeriodArray (values[, dtype, freq, copy])** - Pandas ExtensionArray for storing Period data.
- **PeriodDtype (freq)** - An ExtensionDtype for Period data.
- **Interval** - Immutable object implementing an Interval, a bounded slice-like interval.

- **Interval.closed** - String describing the inclusive side the intervals.
- **Interval.closed_left** - Check if the interval is closed on the left side.
- **Interval.closed_right** - Check if the interval is closed on the right side.
- **Interval.is_empty** - Indicates if an interval is empty, meaning it contains no points.
- **Interval.left** - Left bound for the interval.
- **Interval.length** - Return the length of the Interval.
- **Interval.mid** - Return the midpoint of the Interval.
- **Interval.open_left** - Check if the interval is open on the left side.
- **Interval.open_right** - Check if the interval is open on the right side.
- **Interval.overlaps (other)** - Check whether two Interval objects overlap.
- **Interval.right** - Right bound for the interval.
- **arrays.IntervalArray (data[, closed, dtype, ...])** - Pandas array for interval data that are closed on the same side.
- **IntervalDtype ([subtype, closed])** - An ExtensionDtype for Interval data.
- **arrays.IntegerArray (values, mask[, copy])** - Array of integer (optional missing) values.
- **Int8Dtype ()** - An ExtensionDtype for int8 integer data.
- **Int16Dtype ()** - An ExtensionDtype for int16 integer data.
- **Int32Dtype ()** - An ExtensionDtype for int32 integer data.
- **Int64Dtype ()** - An ExtensionDtype for int64 integer data.
- **UInt8Dtype ()** - An ExtensionDtype for uint8 integer data.
- **UInt16Dtype ()** - An ExtensionDtype for uint16 integer data.
- **UInt32Dtype ()** - An ExtensionDtype for uint32 integer data.
- **UInt64Dtype ()** - An ExtensionDtype for uint64 integer data.
- **arrays.FloatingArray (values, mask[, copy])** - Array of floating (optional missing) values.
- **Float32Dtype ()** - An ExtensionDtype for float32 data.
- **Float64Dtype ()** - An ExtensionDtype for float64 data.
- **CategoricalDtype ([categories, ordered])** - Type for categorical data with the categories and orderedness.
- **CategoricalDtype.categories** - An Index containing the unique categories allowed.
- **CategoricalDtype.ordered** - Whether the categories have an ordered relationship.
- **Categorical (values[, categories, ordered, ...])** - Represent a categorical variable in classic R / S-plus fashion.
- **Categorical.from_codes (codes[, categories, ...])** - Make a Categorical type from codes and categories or dtype.
- **Categorical.dtype** - The CategoricalDtype for this instance.
- **Categorical.categories** - The categories of this categorical.
- **Categorical.ordered** - Whether the categories have an ordered relationship.
- **Categorical.codes** - The category codes of this categorical index.

- **Categorical.__array__ ([dtype, copy])** - The numpy array interface.
- **arrays.SparseArray (data[, sparse_index, ...])** - An ExtensionArray for storing sparse data.
- **SparseDtype ([dtype, fill_value])** - Dtype for data stored in SparseArray .
- **arrays.StringArray (values[, copy])** - Extension array for string data.
- **arrays.ArrowStringArray (values)** - Extension array for string data in a pyarrow.ChunkedArray .
- **StringDtype ([storage])** - Extension dtype for string data.
- **arrays.BooleanArray (values, mask[, copy])** - Array of boolean (True/False) data with missing values.

- **BooleanDtype ()** - Extension dtype for boolean data.

Utilities

- **api.types.union_categoricals (to_union[, ...])** - Combine list-like of Categorical-like, unioning categories.

- **api.types.infer_dtype (value[, skipna])** - Return a string label of the type of a scalar or list-like of values.
- **api.types.pandas_dtype (dtype)** - Convert input into a pandas only dtype object or a numpy dtype object.
- **api.types.is_any_real_numeric_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of a real number dtype.
- **api.types.is_bool_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of a boolean dtype.
- **api.types.is_categorical_dtype (arr_or_dtype)** - (DEPRECATED) Check whether an array-like or dtype is of the Categorical dtype.
- **api.types.is_complex_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of a complex dtype.
- **api.types.is_datetime64_any_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of the datetime64 dtype.
- **api.types.is_datetime64_dtype (arr_or_dtype)** - Check whether an array-like or dtype is of the datetime64 dtype.
- **api.types.is_datetime64_ns_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of the datetime64[ns] dtype.
- **api.types.is_datetime64tz_dtype (arr_or_dtype)** - (DEPRECATED) Check whether an array-like or dtype is of a DatetimeTZDtype dtype.
- **api.types.is_extension_array_dtype (arr_or_dtype)** - Check if an object is a pandas extension array type.
- **api.types.is_float_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of a float dtype.
- **api.types.is_int64_dtype (arr_or_dtype)** - (DEPRECATED) Check whether the provided array or dtype is of the int64 dtype.

- **api.types.is_integer_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of an integer dtype.
- **api.types.is_interval_dtype (arr_or_dtype)** - (DEPRECATED) Check whether an array-like or dtype is of the Interval dtype.
- **api.types.is_numeric_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of a numeric dtype.
- **api.types.is_object_dtype (arr_or_dtype)** - Check whether an array-like or dtype is of the object dtype.
- **api.types.is_period_dtype (arr_or_dtype)** - (DEPRECATED) Check whether an array-like or dtype is of the Period dtype.
- **api.types.is_signed_integer_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of a signed integer dtype.
- **api.types.is_string_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of the string dtype.
- **api.types.is_timedelta64_dtype (arr_or_dtype)** - Check whether an array-like or dtype is of the timedelta64 dtype.
- **api.types.is_timedelta64_ns_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of the timedelta64[ns] dtype.
- **api.types.is_unsigned_integer_dtype (arr_or_dtype)** - Check whether the provided array or dtype is of an unsigned integer dtype.
- **api.types.is_sparse (arr)** - (DEPRECATED) Check whether an array-like is a 1-D pandas sparse array.
- **api.types.is_dict_like (obj)** - Check if the object is dict-like.
- **api.types.is_file_like (obj)** - Check if the object is a file-like object.
- **api.types.is_list_like (obj[, allow_sets])** - Check if the object is list-like.
- **api.types.is_named_tuple (obj)** - Check if the object is a named tuple.
- **api.types.is_iterator (obj)** - Check if the object is an iterator.
- **api.types.is_bool (obj)** - Return True if given object is boolean.
- **api.types.is_complex (obj)** - Return True if given object is complex.
- **api.types.is_float (obj)** - Return True if given object is float.
- **api.types.is_hashable (obj)** - Return True if hash(obj) will succeed, False otherwise.
- **api.types.is_integer (obj)** - Return True if given object is integer.
- **api.types.is_interval (obj)** -
- **api.types.is_number (obj)** - Check if the object is a number.
- **api.types.is_re (obj)** - Check if the object is a regex pattern instance.
- **api.types.is_re_compilable (obj)** - Check if the object can be compiled into a regex pattern instance.

- **api.types.is_scalar (val)** - Return True if given object is scalar.

Index objects

Index

- **Index ([data, dtype, copy, name, tupleize_cols])** - Immutable sequence used for indexing and alignment.
- **Index.values** - Return an array representing the data in the Index.
- **Index.is_monotonic_increasing** - Return a boolean if the values are equal or increasing.
- **Index.is_monotonic_decreasing** - Return a boolean if the values are equal or decreasing.
- **Index.is_unique** - Return if the index has unique values.
- **Index.has_duplicates** - Check if the Index has duplicate values.
- **Index.hasnans** - Return True if there are any NaNs.
- **Index.dtype** - Return the dtype object of the underlying data.
- **Index.inferred_type** - Return a string of the type inferred from the values.
- **Index.shape** - Return a tuple of the shape of the underlying data.
- **Index.name** - Return Index or MultiIndex name.
- **Index.names** -
- **Index.nbytes** - Return the number of bytes in the underlying data.
- **Index.ndim** - Number of dimensions of the underlying data, by definition 1.
- **Index.size** - Return the number of elements in the underlying data.
- **Index.empty** -
- **Index.T** - Return the transpose, which is by definition self.
- **Index.memory_usage ([deep])** - Memory usage of the values.
- **Index.all (*args, **kwargs)** - Return whether all elements are Truthy.
- **Index.any (*args, **kwargs)** - Return whether any element is Truthy.
- **Index.argmin ([axis, skipna])** - Return int position of the smallest value in the Series.
- **Index.argmax ([axis, skipna])** - Return int position of the largest value in the Series.
- **Index.copy ([name, deep])** - Make a copy of this object.
- **Index.delete (loc)** - Make new Index with passed location(-s) deleted.
- **Index.drop (labels[, errors])** - Make new Index with passed list of labels deleted.
- **Index.drop_duplicates (*[, keep])** - Return Index with duplicate values removed.
- **Index.duplicated ([keep])** - Indicate duplicate index values.

- **Index.equals (other)** - Determine if two Index object are equal.
- **Index.factorize ([sort, use_na_sentinel])** - Encode the object as an enumerated type or categorical variable.
- **Index.identical (other)** - Similar to equals, but checks that object attributes and types are also equal.
- **Index.insert (loc, item)** - Make new Index inserting new item at location.
- **Index.is_ (other)** - More flexible, faster check like is but that works through views.
- **Index.is_boolean ()** - (DEPRECATED) Check if the Index only consists of booleans.
- **Index.is_categorical ()** - (DEPRECATED) Check if the Index holds categorical data.
- **Index.is_floating ()** - (DEPRECATED) Check if the Index is a floating type.
- **Index.is_integer ()** - (DEPRECATED) Check if the Index only consists of integers.
- **Index.is_interval ()** - (DEPRECATED) Check if the Index holds Interval objects.
- **Index.is_numeric ()** - (DEPRECATED) Check if the Index only consists of numeric data.
- **Index.is_object ()** - (DEPRECATED) Check if the Index is of the object dtype.
- **Index.min ([axis, skipna])** - Return the minimum value of the Index.
- **Index.max ([axis, skipna])** - Return the maximum value of the Index.
- **Index.reindex (target[, method, level, ...])** - Create index with target's values.
- **Index.rename (name, *[, inplace])** - Alter Index or MultiIndex name.
- **Index.repeat (repeats[, axis])** - Repeat elements of a Index.
- **Index.where (cond[, other])** - Replace values where the condition is False.
- **Index.take (indices[, axis, allow_fill, ...])** - Return a new Index of the values selected by the indices.
- **Index.putmask (mask, value)** - Return a new Index of the values set with the mask.
- **Index.unique ([level])** - Return unique values in the index.
- **Index.nunique ([dropna])** - Return number of unique elements in the object.
- **Index.value_counts ([normalize, sort, ...])** - Return a Series containing counts of unique values.
- **Index.set_names (names, *[, level, inplace])** - Set Index or MultiIndex name.
- **Index.droplevel ([level])** - Return index with requested level(s) removed.
- **Index.fillna ([value, downcast])** - Fill NA/NaN values with the specified value.
- **Index.dropna ([how])** - Return Index without NA/NaN values.
- **Index.isna ()** - Detect missing values.
- **Index.notna ()** - Detect existing (non-missing) values.

- **Index.astype (dtype[, copy])** - Create an Index with values cast to dtypes.
- **Index.item ()** - Return the first element of the underlying data as a Python scalar.
- **Index.map (mapper[, na_action])** - Map values using an input mapping or function.
- **Index.ravel ([order])** - Return a view on self.
- **Index.to_list ()** - Return a list of the values.
- **Index.to_series ([index, name])** - Create a Series with both index and values equal to the index keys.
- **Index.to_frame ([index, name])** - Create a DataFrame with a column containing the Index.
- **Index.view ([cls])** -
- **Index.argsort (*args, **kwargs)** - Return the integer indices that would sort the index.
- **Index.searchsorted (value[, side, sorter])** - Find indices where elements should be inserted to maintain order.
- **Index.sort_values (*[, return_indexer, ...])** - Return a sorted copy of the index.
- **Index.shift ([periods, freq])** - Shift index by desired number of time frequency increments.
- **Index.append (other)** - Append a collection of Index options together.
- **Index.join (other, *[, how, level, ...])** - Compute join_index and indexers to conform data structures to the new index.
- **Index.intersection (other[, sort])** - Form the intersection of two Index objects.
- **Index.union (other[, sort])** - Form the union of two Index objects.
- **Index.difference (other[, sort])** - Return a new Index with elements of index not in other .
- **Index.symmetric_difference (other[, ...])** - Compute the symmetric difference of two Index objects.
- **Index.asof (label)** - Return the label from the index, or, if not present, the previous one.
- **Index.asof_locs (where, mask)** - Return the locations (indices) of labels in the index.
- **Index.get_indexer (target[, method, limit, ...])** - Compute indexer and mask for new index given the current index.
- **Index.get_indexer_for (target)** - Guaranteed return of an indexer even when non-unique.
- **Index.get_indexer_non_unique (target)** - Compute indexer and mask for new index given the current index.
- **Index.get_level_values (level)** - Return an Index of values for requested level.
- **Index.get_loc (key)** - Get integer location, slice or boolean mask for requested label.
- **Index.get_slice_bound (label, side)** - Calculate slice bound that corresponds to given label.
- **Index.isin (values[, level])** - Return a boolean array where the index values are in values .
- **Index.slice_indexer ([start, end, step])** - Compute the slice indexer for input labels and step.

- **Index.slice_locs ([start, end, step])** - Compute slice locations for input labels.

Numeric Index

- **RangeIndex ([start, stop, step, dtype, copy, ...])** - Immutable Index implementing a monotonic integer range.

- **RangeIndex.start** - The value of the start parameter (0 if this was not supplied).
- **RangeIndex.stop** - The value of the stop parameter.
- **RangeIndex.step** - The value of the step parameter (1 if this was not supplied).

- **RangeIndex.from_range (data[, name, dtype])** - Create pandas.RangeIndex from a range object.

CategoricalIndex

- **CategoricalIndex ([data, categories, ...])** - Index based on an underlying Categorical .

- **CategoricalIndex.codes** - The category codes of this categorical index.
- **CategoricalIndex.categories** - The categories of this categorical.
- **CategoricalIndex.ordered** - Whether the categories have an ordered relationship.
- **CategoricalIndex.rename_categories (*args, ...)** - Rename categories.
- **CategoricalIndex.reorder_categories (*args, ...)** - Reorder categories as specified in new_categories.
- **CategoricalIndex.add_categories (*args, **kwargs)** - Add new categories.
- **CategoricalIndex.remove_categories (*args, ...)** - Remove the specified categories.
- **CategoricalIndex.remove_unused_categories (...)** - Remove categories which are not used.
- **CategoricalIndex.set_categories (*args, **kwargs)** - Set the categories to the specified new categories.
- **CategoricalIndex.as_ordered (*args, **kwargs)** - Set the Categorical to be ordered.
- **CategoricalIndex.as_unordered (*args, **kwargs)** - Set the Categorical to be unordered.
- **CategoricalIndex.map (mapper[, na_action])** - Map values using input an input mapping or function.

- **CategoricalIndex.equals (other)** - Determine if two CategoricalIndex objects contain the same elements.

IntervalIndex

- **IntervalIndex (data[, closed, dtype, copy, ...])** - Immutable index of intervals that are closed on the same side.

- **IntervalIndex.from_arrays (left, right[, ...])** - Construct from two arrays defining the left and right bounds.
- **IntervalIndex.from_tuples (data[, closed, ...])** - Construct an IntervalIndex from an array-like of tuples.
- **IntervalIndex.from_breaks (breaks[, closed, ...])** - Construct an IntervalIndex from an array of splits.
- **IntervalIndex.left** -
- **IntervalIndex.right** -
- **IntervalIndex.mid** -
- **IntervalIndex.closed** - String describing the inclusive side the intervals.
- **IntervalIndex.length** -
- **IntervalIndex.values** - Return an array representing the data in the Index.
- **IntervalIndex.is_empty** - Indicates if an interval is empty, meaning it contains no points.
- **IntervalIndex.is_non_overlapping_monotonic** - Return a boolean whether the IntervalArray is non-overlapping and monotonic.
- **IntervalIndex.is_overlapping** - Return True if the IntervalIndex has overlapping intervals, else False.
- **IntervalIndex.get_loc (key)** - Get integer location, slice or boolean mask for requested label.
- **IntervalIndex.get_indexer (target[, method, ...])** - Compute indexer and mask for new index given the current index.
- **IntervalIndex.set_closed (*args, **kwargs)** - Return an identical IntervalArray closed on the specified side.
- **IntervalIndex.contains (*args, **kwargs)** - Check elementwise if the Intervals contain the value.
- **IntervalIndex.overlaps (*args, **kwargs)** - Check elementwise if an Interval overlaps the values in the IntervalArray.

- **IntervalIndex.to_tuples (*args, **kwargs)** - Return an ndarray (if self is IntervalArray) or Index (if self is IntervalIndex) of tuples of the form (left, right).

MultiIndex

- **MultiIndex ([levels, codes, sortorder, ...])** - A multi-level, or hierarchical, index object for pandas objects.

- **MultiIndex.from_arrays (arrays[, sortorder, ...])** - Convert arrays to MultiIndex.
- **MultiIndex.from_tuples (tuples[, sortorder, ...])** - Convert list of tuples to MultiIndex.
- **MultiIndex.from_product (iterables[, ...])** - Make a MultiIndex from the cartesian product of multiple iterables.

- **MultiIndex.from_frame (df[, sortorder, names])** - Make a MultiIndex from a DataFrame.
- **MultiIndex.names** - Names of levels in MultiIndex.
- **MultiIndex.levels** - Levels of the MultiIndex.
- **MultiIndex.codes** -
- **MultiIndex.nlevels** - Integer number of levels in this MultiIndex.
- **MultiIndex.levshape** - A tuple with the length of each level.
- **MultiIndex.dtypes** - Return the dtypes as a Series for the underlying MultiIndex.
- **MultiIndex.set_levels (levels, *[, level, ...])** - Set new levels on MultiIndex.
- **MultiIndex.set_codes (codes, *[, level, ...])** - Set new codes on MultiIndex.
- **MultiIndex.to_flat_index ()** - Convert a MultiIndex to an Index of Tuples containing the level values.
- **MultiIndex.to_frame ([index, name, ...])** - Create a DataFrame with the levels of the MultiIndex as columns.
- **MultiIndex.sortlevel ([level, ascending, ...])** - Sort MultiIndex at the requested level.
- **MultiIndex.droplevel ([level])** - Return index with requested level(s) removed.
- **MultiIndex.swaplevel ([i, j])** - Swap level i with level j.
- **MultiIndex.reorder_levels (order)** - Rearrange levels using input order.
- **MultiIndex.remove_unused_levels ()** - Create new MultiIndex from current that removes unused levels.
- **MultiIndex.drop (codes[, level, errors])** - Make a new pandas.MultiIndex with the passed list of codes deleted.
- **MultiIndex.copy ([names, deep, name])** - Make a copy of this object.
- **MultiIndex.append (other)** - Append a collection of Index options together.
- **MultiIndex.truncate ([before, after])** - Slice index between two labels / tuples, return new MultiIndex.
- **MultiIndex.get_loc (key)** - Get location for a label or a tuple of labels.
- **MultiIndex.get_locs (seq)** - Get location for a sequence of labels.
- **MultiIndex.get_loc_level (key[, level, ...])** - Get location and sliced index for requested label(s)/level(s).
- **MultiIndex.get_indexer (target[, method, ...])** - Compute indexer and mask for new index given the current index.
- **MultiIndex.get_level_values (level)** - Return vector of label values for requested level.

- **IndexSlice** - Create an object to more easily perform multi-index slicing.

DatetimeIndex

- **DatetimeIndex ([data, freq, tz, normalize, ...])** - Immutable ndarray-like of datetime64 data.

- **DatetimeIndex.year** - The year of the datetime.
- **DatetimeIndex.month** - The month as January\=1, December\=12.
- **DatetimeIndex.day** - The day of the datetime.
- **DatetimeIndex.hour** - The hours of the datetime.
- **DatetimeIndex.minute** - The minutes of the datetime.
- **DatetimeIndex.second** - The seconds of the datetime.
- **DatetimeIndex.microsecond** - The microseconds of the datetime.
- **DatetimeIndex.nanosecond** - The nanoseconds of the datetime.
- **DatetimeIndex.date** - Returns numpy array of python datetime.date objects.
- **DatetimeIndex.time** - Returns numpy array of datetime.time objects.
- **DatetimeIndex.timetz** - Returns numpy array of datetime.time objects with timezones.
- **DatetimeIndex.dayofyear** - The ordinal day of the year.
- **DatetimeIndex.day_of_year** - The ordinal day of the year.
- **DatetimeIndex.dayofweek** - The day of the week with Monday\=0, Sunday\=6.
- **DatetimeIndex.day_of_week** - The day of the week with Monday\=0, Sunday\=6.
- **DatetimeIndex.weekday** - The day of the week with Monday\=0, Sunday\=6.
- **DatetimeIndex.quarter** - The quarter of the date.
- **DatetimeIndex.tz** - Return the timezone.
- **DatetimeIndex.freq** -
- **DatetimeIndex.freqstr** - Return the frequency object as a string if it's set, otherwise None.
- **DatetimeIndex.is_month_start** - Indicates whether the date is the first day of the month.
- **DatetimeIndex.is_month_end** - Indicates whether the date is the last day of the month.
- **DatetimeIndex.is_quarter_start** - Indicator for whether the date is the first day of a quarter.
- **DatetimeIndex.is_quarter_end** - Indicator for whether the date is the last day of a quarter.
- **DatetimeIndex.is_year_start** - Indicate whether the date is the first day of a year.
- **DatetimeIndex.is_year_end** - Indicate whether the date is the last day of the year.
- **DatetimeIndex.is_leap_year** - Boolean indicator if the date belongs to a leap year.
- **DatetimeIndex.inferred_freq** - Tries to return a string representing a frequency generated by infer_freq.
- **DatetimeIndex.indexer_at_time (time[, asof])** - Return index locations of values at particular time of day.
- **DatetimeIndex.indexer_between_time (...[, ...])** - Return index locations of values between particular times of day.
- **DatetimeIndex.normalize (*args, **kwargs)** - Convert times to midnight.
- **DatetimeIndex.strftime (date_format)** - Convert to Index using specified date_format.
- **DatetimeIndex.snap ([freq])** - Snap time stamps to nearest occurring frequency.

- **DatetimeIndex.tz_convert (tz)** - Convert tz-aware Datetime Array/Index from one time zone to another.
- **DatetimeIndex.tz_localize (tz[, ambiguous, ...])** - Localize tz-naive Datetime Array/Index to tz-aware Datetime Array/ Index.
- **DatetimeIndex.round (*args, **kwargs)** - Perform round operation on the data to the specified freq .
- **DatetimeIndex.floor (*args, **kwargs)** - Perform floor operation on the data to the specified freq .
- **DatetimeIndex.ceil (*args, **kwargs)** - Perform ceil operation on the data to the specified freq .
- **DatetimeIndex.month_name (*args, **kwargs)** - Return the month names with specified locale.
- **DatetimeIndex.day_name (*args, **kwargs)** - Return the day names with specified locale.
- **DatetimeIndex.as_unit (*args, **kwargs)** - Convert to a dtype with the given unit resolution.
- **DatetimeIndex.to_period (*args, **kwargs)** - Cast to PeriodArray/PeriodIndex at a particular frequency.
- **DatetimeIndex.to_pydatetime (*args, **kwargs)** - Return an ndarray of datetime.datetime objects.
- **DatetimeIndex.to_series ([index, name])** - Create a Series with both index and values equal to the index keys.
- **DatetimeIndex.to_frame ([index, name])** - Create a DataFrame with a column containing the Index.
- **DatetimeIndex.mean (*[, skipna, axis])** - Return the mean value of the Array.

- **DatetimeIndex.std (*args, **kwargs)** - Return sample standard deviation over requested axis.

TimedeltaIndex

- **TimedeltaIndex ([data, unit, freq, closed, ...])** - Immutable Index of timedelta64 data.

- **TimedeltaIndex.days** - Number of days for each element.
- **TimedeltaIndex.seconds** - Number of seconds (>\= 0 and less than 1 day) for each element.
- **TimedeltaIndex.microseconds** - Number of microseconds (>\= 0 and less than 1 second) for each element.
- **TimedeltaIndex.nanoseconds** - Number of nanoseconds (>\= 0 and less than 1 microsecond) for each element.
- **TimedeltaIndex.components** - Return a DataFrame of the individual resolution components of the Timedeltas.
- **TimedeltaIndex.inferred_freq** - Tries to return a string representing a frequency generated by infer_freq.
- **TimedeltaIndex.as_unit (unit)** - Convert to a dtype with the given unit resolution.
- **TimedeltaIndex.to_pytimedelta (*args, **kwargs)** - Return an ndarray of datetime.timedelta objects.
- **TimedeltaIndex.to_series ([index, name])** - Create a Series with both index and values equal to the index keys.

- **TimedeltaIndex.round (*args, **kwargs)** - Perform round operation on the data to the specified freq .
- **TimedeltaIndex.floor (*args, **kwargs)** - Perform floor operation on the data to the specified freq .
- **TimedeltaIndex.ceil (*args, **kwargs)** - Perform ceil operation on the data to the specified freq .
- **TimedeltaIndex.to_frame ([index, name])** - Create a DataFrame with a column containing the Index.

- **TimedeltaIndex.mean (*[, skipna, axis])** - Return the mean value of the Array.

PeriodIndex

- **PeriodIndex ([data, ordinal, freq, dtype, ...])** - Immutable ndarray holding ordinal values indicating regular periods in time.

- **PeriodIndex.day** - The days of the period.
- **PeriodIndex.dayofweek** - The day of the week with Monday\=0, Sunday\=6.
- **PeriodIndex.day_of_week** - The day of the week with Monday\=0, Sunday\=6.
- **PeriodIndex.dayofyear** - The ordinal day of the year.
- **PeriodIndex.day_of_year** - The ordinal day of the year.
- **PeriodIndex.days_in_month** - The number of days in the month.
- **PeriodIndex.daysinmonth** - The number of days in the month.
- **PeriodIndex.end_time** - Get the Timestamp for the end of the period.
- **PeriodIndex.freq** -
- **PeriodIndex.freqstr** - Return the frequency object as a string if it's set, otherwise None.
- **PeriodIndex.hour** - The hour of the period.
- **PeriodIndex.is_leap_year** - Logical indicating if the date belongs to a leap year.
- **PeriodIndex.minute** - The minute of the period.
- **PeriodIndex.month** - The month as January\=1, December\=12.
- **PeriodIndex.quarter** - The quarter of the date.
- **PeriodIndex.qyear** -
- **PeriodIndex.second** - The second of the period.
- **PeriodIndex.start_time** - Get the Timestamp for the start of the period.
- **PeriodIndex.week** - The week ordinal of the year.
- **PeriodIndex.weekday** - The day of the week with Monday\=0, Sunday\=6.
- **PeriodIndex.weekofyear** - The week ordinal of the year.
- **PeriodIndex.year** - The year of the period.
- **PeriodIndex.asfreq ([freq, how])** - Convert the PeriodArray to the specified frequency freq .
- **PeriodIndex.strftime (*args, **kwargs)** - Convert to Index using specified date_format.
- **PeriodIndex.to_timestamp ([freq, how])** - Cast to DatetimeArray/Index.

- **PeriodIndex.from_fields (*[, year, quarter, ...])** -
- **PeriodIndex.from_ordinals (ordinals, *, freq)** -

Date offsets

DateOffset

- **DateOffset** - Standard kind of date increment used for a date range.
- **DateOffset.freqstr** - Return a string representing the frequency.
- **DateOffset.kwds** - Return a dict of extra parameters for the offset.
- **DateOffset.name** - Return a string representing the base frequency.
- **DateOffset.nanos** -
- **DateOffset.normalize** -
- **DateOffset.rule_code** -
- **DateOffset.n** -
- **DateOffset.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **DateOffset.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **DateOffset.copy ()** - Return a copy of the frequency.
- **DateOffset.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **DateOffset.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **DateOffset.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **DateOffset.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **DateOffset.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **DateOffset.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **DateOffset.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **DateOffset.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

  ## BusinessDay

- **BusinessDay** - DateOffset subclass representing possibly n business days.

- **BDay** - alias of BusinessDay
- **BusinessDay.freqstr** - Return a string representing the frequency.

- **BusinessDay.kwds** - Return a dict of extra parameters for the offset.
- **BusinessDay.name** - Return a string representing the base frequency.
- **BusinessDay.nanos** -
- **BusinessDay.normalize** -
- **BusinessDay.rule_code** -
- **BusinessDay.n** -
- **BusinessDay.weekmask** -
- **BusinessDay.holidays** -
- **BusinessDay.calendar** -
- **BusinessDay.copy ()** - Return a copy of the frequency.
- **BusinessDay.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **BusinessDay.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **BusinessDay.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **BusinessDay.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **BusinessDay.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **BusinessDay.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **BusinessDay.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **BusinessDay.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

BusinessHour

- **BusinessHour** - DateOffset subclass representing possibly n business hours.

- **BusinessHour.freqstr** - Return a string representing the frequency.
- **BusinessHour.kwds** - Return a dict of extra parameters for the offset.
- **BusinessHour.name** - Return a string representing the base frequency.
- **BusinessHour.nanos** -
- **BusinessHour.normalize** -
- **BusinessHour.rule_code** -
- **BusinessHour.n** -
- **BusinessHour.start** -
- **BusinessHour.end** -
- **BusinessHour.weekmask** -
- **BusinessHour.holidays** -
- **BusinessHour.calendar** -
- **BusinessHour.copy ()** - Return a copy of the frequency.
- **BusinessHour.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).

- **BusinessHour.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **BusinessHour.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **BusinessHour.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **BusinessHour.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **BusinessHour.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **BusinessHour.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **BusinessHour.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

CustomBusinessDay

- **CustomBusinessDay** - DateOffset subclass representing possibly n custom business days.

- **CDay** - alias of CustomBusinessDay
- **CustomBusinessDay.freqstr** - Return a string representing the frequency.
- **CustomBusinessDay.kwds** - Return a dict of extra parameters for the offset.
- **CustomBusinessDay.name** - Return a string representing the base frequency.
- **CustomBusinessDay.nanos** -
- **CustomBusinessDay.normalize** -
- **CustomBusinessDay.rule_code** -
- **CustomBusinessDay.n** -
- **CustomBusinessDay.weekmask** -
- **CustomBusinessDay.calendar** -
- **CustomBusinessDay.holidays** -
- **CustomBusinessDay.copy ()** - Return a copy of the frequency.
- **CustomBusinessDay.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **CustomBusinessDay.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **CustomBusinessDay.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **CustomBusinessDay.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **CustomBusinessDay.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **CustomBusinessDay.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **CustomBusinessDay.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **CustomBusinessDay.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

CustomBusinessHour

- **CustomBusinessHour** - DateOffset subclass representing possibly n custom business days.

- **CustomBusinessHour.freqstr** - Return a string representing the frequency.
- **CustomBusinessHour.kwds** - Return a dict of extra parameters for the offset.
- **CustomBusinessHour.name** - Return a string representing the base frequency.
- **CustomBusinessHour.nanos** -
- **CustomBusinessHour.normalize** -
- **CustomBusinessHour.rule_code** -
- **CustomBusinessHour.n** -
- **CustomBusinessHour.weekmask** -
- **CustomBusinessHour.calendar** -
- **CustomBusinessHour.holidays** -
- **CustomBusinessHour.start** -
- **CustomBusinessHour.end** -
- **CustomBusinessHour.copy ()** - Return a copy of the frequency.
- **CustomBusinessHour.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **CustomBusinessHour.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **CustomBusinessHour.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **CustomBusinessHour.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **CustomBusinessHour.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **CustomBusinessHour.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **CustomBusinessHour.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **CustomBusinessHour.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

MonthEnd

- **MonthEnd** - DateOffset of one month end.

- **MonthEnd.freqstr** - Return a string representing the frequency.
- **MonthEnd.kwds** - Return a dict of extra parameters for the offset.
- **MonthEnd.name** - Return a string representing the base frequency.
- **MonthEnd.nanos** -
- **MonthEnd.normalize** -
- **MonthEnd.rule_code** -

- **MonthEnd.n** -
- **MonthEnd.copy ()** - Return a copy of the frequency.
- **MonthEnd.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **MonthEnd.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **MonthEnd.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **MonthEnd.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **MonthEnd.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **MonthEnd.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **MonthEnd.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **MonthEnd.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

MonthBegin

- **MonthBegin** - DateOffset of one month at beginning.

- **MonthBegin.freqstr** - Return a string representing the frequency.
- **MonthBegin.kwds** - Return a dict of extra parameters for the offset.
- **MonthBegin.name** - Return a string representing the base frequency.
- **MonthBegin.nanos** -
- **MonthBegin.normalize** -
- **MonthBegin.rule_code** -
- **MonthBegin.n** -
- **MonthBegin.copy ()** - Return a copy of the frequency.
- **MonthBegin.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **MonthBegin.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **MonthBegin.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **MonthBegin.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **MonthBegin.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **MonthBegin.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **MonthBegin.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **MonthBegin.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

BusinessMonthEnd

- **BusinessMonthEnd** - DateOffset increments between the last business day of the month.

- **BMonthEnd** - alias of BusinessMonthEnd
- **BusinessMonthEnd.freqstr** - Return a string representing the frequency.
- **BusinessMonthEnd.kwds** - Return a dict of extra parameters for the offset.
- **BusinessMonthEnd.name** - Return a string representing the base frequency.
- **BusinessMonthEnd.nanos** -
- **BusinessMonthEnd.normalize** -
- **BusinessMonthEnd.rule_code** -
- **BusinessMonthEnd.n** -
- **BusinessMonthEnd.copy ()** - Return a copy of the frequency.
- **BusinessMonthEnd.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **BusinessMonthEnd.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **BusinessMonthEnd.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **BusinessMonthEnd.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **BusinessMonthEnd.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **BusinessMonthEnd.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **BusinessMonthEnd.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **BusinessMonthEnd.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

BusinessMonthBegin

- **BusinessMonthBegin** - DateOffset of one month at the first business day.

- **BMonthBegin** - alias of BusinessMonthBegin
- **BusinessMonthBegin.freqstr** - Return a string representing the frequency.
- **BusinessMonthBegin.kwds** - Return a dict of extra parameters for the offset.
- **BusinessMonthBegin.name** - Return a string representing the base frequency.
- **BusinessMonthBegin.nanos** -
- **BusinessMonthBegin.normalize** -
- **BusinessMonthBegin.rule_code** -
- **BusinessMonthBegin.n** -
- **BusinessMonthBegin.copy ()** - Return a copy of the frequency.

- **BusinessMonthBegin.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **BusinessMonthBegin.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **BusinessMonthBegin.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **BusinessMonthBegin.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **BusinessMonthBegin.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **BusinessMonthBegin.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **BusinessMonthBegin.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **BusinessMonthBegin.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

CustomBusinessMonthEnd

- **CustomBusinessMonthEnd** - DateOffset subclass representing custom business month(s).

- **CBMonthEnd** - alias of CustomBusinessMonthEnd
- **CustomBusinessMonthEnd.freqstr** - Return a string representing the frequency.
- **CustomBusinessMonthEnd.kwds** - Return a dict of extra parameters for the offset.
- **CustomBusinessMonthEnd.m_offset** -
- **CustomBusinessMonthEnd.name** - Return a string representing the base frequency.
- **CustomBusinessMonthEnd.nanos** -
- **CustomBusinessMonthEnd.normalize** -
- **CustomBusinessMonthEnd.rule_code** -
- **CustomBusinessMonthEnd.n** -
- **CustomBusinessMonthEnd.weekmask** -
- **CustomBusinessMonthEnd.calendar** -
- **CustomBusinessMonthEnd.holidays** -
- **CustomBusinessMonthEnd.copy ()** - Return a copy of the frequency.
- **CustomBusinessMonthEnd.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **CustomBusinessMonthEnd.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **CustomBusinessMonthEnd.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **CustomBusinessMonthEnd.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **CustomBusinessMonthEnd.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **CustomBusinessMonthEnd.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.

- **CustomBusinessMonthEnd.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **CustomBusinessMonthEnd.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

CustomBusinessMonthBegin

- **CustomBusinessMonthBegin** - DateOffset subclass representing custom business month(s).

- **CBMonthBegin** - alias of CustomBusinessMonthBegin
- **CustomBusinessMonthBegin.freqstr** - Return a string representing the frequency.
- **CustomBusinessMonthBegin.kwds** - Return a dict of extra parameters for the offset.
- **CustomBusinessMonthBegin.m_offset** -
- **CustomBusinessMonthBegin.name** - Return a string representing the base frequency.
- **CustomBusinessMonthBegin.nanos** -
- **CustomBusinessMonthBegin.normalize** -
- **CustomBusinessMonthBegin.rule_code** -
- **CustomBusinessMonthBegin.n** -
- **CustomBusinessMonthBegin.weekmask** -
- **CustomBusinessMonthBegin.calendar** -
- **CustomBusinessMonthBegin.holidays** -
- **CustomBusinessMonthBegin.copy ()** - Return a copy of the frequency.
- **CustomBusinessMonthBegin.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **CustomBusinessMonthBegin.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **CustomBusinessMonthBegin.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **CustomBusinessMonthBegin.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **CustomBusinessMonthBegin.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **CustomBusinessMonthBegin.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **CustomBusinessMonthBegin.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **CustomBusinessMonthBegin.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

SemiMonthEnd

- **SemiMonthEnd** - Two DateOffset's per month repeating on the last day of the month & day_of_month.

- **SemiMonthEnd.freqstr** - Return a string representing the frequency.
- **SemiMonthEnd.kwds** - Return a dict of extra parameters for the offset.
- **SemiMonthEnd.name** - Return a string representing the base frequency.
- **SemiMonthEnd.nanos** -
- **SemiMonthEnd.normalize** -
- **SemiMonthEnd.rule_code** -
- **SemiMonthEnd.n** -
- **SemiMonthEnd.day_of_month** -
- **SemiMonthEnd.copy ()** - Return a copy of the frequency.
- **SemiMonthEnd.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **SemiMonthEnd.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **SemiMonthEnd.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **SemiMonthEnd.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **SemiMonthEnd.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **SemiMonthEnd.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **SemiMonthEnd.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **SemiMonthEnd.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

SemiMonthBegin

- **SemiMonthBegin** - Two DateOffset's per month repeating on the first day of the month & day_of_month.

- **SemiMonthBegin.freqstr** - Return a string representing the frequency.
- **SemiMonthBegin.kwds** - Return a dict of extra parameters for the offset.
- **SemiMonthBegin.name** - Return a string representing the base frequency.
- **SemiMonthBegin.nanos** -
- **SemiMonthBegin.normalize** -
- **SemiMonthBegin.rule_code** -
- **SemiMonthBegin.n** -
- **SemiMonthBegin.day_of_month** -
- **SemiMonthBegin.copy ()** - Return a copy of the frequency.
- **SemiMonthBegin.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **SemiMonthBegin.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **SemiMonthBegin.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.

- **SemiMonthBegin.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **SemiMonthBegin.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **SemiMonthBegin.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **SemiMonthBegin.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **SemiMonthBegin.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Week

- **Week** - Weekly offset.

- **Week.freqstr** - Return a string representing the frequency.
- **Week.kwds** - Return a dict of extra parameters for the offset.
- **Week.name** - Return a string representing the base frequency.
- **Week.nanos** -
- **Week.normalize** -
- **Week.rule_code** -
- **Week.n** -
- **Week.weekday** -
- **Week.copy ()** - Return a copy of the frequency.
- **Week.is_anchored ()** - Return boolean whether the frequency is a unit frequency (n\=1).
- **Week.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Week.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Week.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Week.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Week.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Week.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Week.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

WeekOfMonth

- **WeekOfMonth** - Describes monthly dates like "the Tuesday of the 2nd week of each month".

- **WeekOfMonth.freqstr** - Return a string representing the frequency.
- **WeekOfMonth.kwds** - Return a dict of extra parameters for the offset.

- **WeekOfMonth.name** - Return a string representing the base frequency.
- **WeekOfMonth.nanos** -
- **WeekOfMonth.normalize** -
- **WeekOfMonth.rule_code** -
- **WeekOfMonth.n** -
- **WeekOfMonth.week** -
- **WeekOfMonth.copy ()** - Return a copy of the frequency.
- **WeekOfMonth.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **WeekOfMonth.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **WeekOfMonth.weekday** -
- **WeekOfMonth.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **WeekOfMonth.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **WeekOfMonth.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **WeekOfMonth.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **WeekOfMonth.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **WeekOfMonth.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

LastWeekOfMonth

- **LastWeekOfMonth** - Describes monthly dates in last week of month.

- **LastWeekOfMonth.freqstr** - Return a string representing the frequency.
- **LastWeekOfMonth.kwds** - Return a dict of extra parameters for the offset.
- **LastWeekOfMonth.name** - Return a string representing the base frequency.
- **LastWeekOfMonth.nanos** -
- **LastWeekOfMonth.normalize** -
- **LastWeekOfMonth.rule_code** -
- **LastWeekOfMonth.n** -
- **LastWeekOfMonth.weekday** -
- **LastWeekOfMonth.week** -
- **LastWeekOfMonth.copy ()** - Return a copy of the frequency.
- **LastWeekOfMonth.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **LastWeekOfMonth.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **LastWeekOfMonth.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **LastWeekOfMonth.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.

- **LastWeekOfMonth.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **LastWeekOfMonth.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **LastWeekOfMonth.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **LastWeekOfMonth.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

BQuarterEnd

- **BQuarterEnd** - DateOffset increments between the last business day of each Quarter.

- **BQuarterEnd.freqstr** - Return a string representing the frequency.
- **BQuarterEnd.kwds** - Return a dict of extra parameters for the offset.
- **BQuarterEnd.name** - Return a string representing the base frequency.
- **BQuarterEnd.nanos** -
- **BQuarterEnd.normalize** -
- **BQuarterEnd.rule_code** -
- **BQuarterEnd.n** -
- **BQuarterEnd.startingMonth** -
- **BQuarterEnd.copy ()** - Return a copy of the frequency.
- **BQuarterEnd.is_anchored ()** - Return boolean whether the frequency is a unit frequency (n\=1).
- **BQuarterEnd.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **BQuarterEnd.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **BQuarterEnd.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **BQuarterEnd.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **BQuarterEnd.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **BQuarterEnd.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **BQuarterEnd.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

BQuarterBegin

- **BQuarterBegin** - DateOffset increments between the first business day of each Quarter.

- **BQuarterBegin.freqstr** - Return a string representing the frequency.

- **BQuarterBegin.kwds** - Return a dict of extra parameters for the offset.
- **BQuarterBegin.name** - Return a string representing the base frequency.
- **BQuarterBegin.nanos** -
- **BQuarterBegin.normalize** -
- **BQuarterBegin.rule_code** -
- **BQuarterBegin.n** -
- **BQuarterBegin.startingMonth** -
- **BQuarterBegin.copy ()** - Return a copy of the frequency.
- **BQuarterBegin.is_anchored ()** - Return boolean whether the frequency is a unit frequency (n\=1).
- **BQuarterBegin.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **BQuarterBegin.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **BQuarterBegin.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **BQuarterBegin.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **BQuarterBegin.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **BQuarterBegin.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **BQuarterBegin.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

QuarterEnd

- **QuarterEnd** - DateOffset increments between Quarter end dates.

- **QuarterEnd.freqstr** - Return a string representing the frequency.
- **QuarterEnd.kwds** - Return a dict of extra parameters for the offset.
- **QuarterEnd.name** - Return a string representing the base frequency.
- **QuarterEnd.nanos** -
- **QuarterEnd.normalize** -
- **QuarterEnd.rule_code** -
- **QuarterEnd.n** -
- **QuarterEnd.startingMonth** -
- **QuarterEnd.copy ()** - Return a copy of the frequency.
- **QuarterEnd.is_anchored ()** - Return boolean whether the frequency is a unit frequency (n\=1).
- **QuarterEnd.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **QuarterEnd.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **QuarterEnd.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.

- **QuarterEnd.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **QuarterEnd.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **QuarterEnd.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **QuarterEnd.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

QuarterBegin

- **QuarterBegin** - DateOffset increments between Quarter start dates.

- **QuarterBegin.freqstr** - Return a string representing the frequency.
- **QuarterBegin.kwds** - Return a dict of extra parameters for the offset.
- **QuarterBegin.name** - Return a string representing the base frequency.
- **QuarterBegin.nanos** -
- **QuarterBegin.normalize** -
- **QuarterBegin.rule_code** -
- **QuarterBegin.n** -
- **QuarterBegin.startingMonth** -
- **QuarterBegin.copy ()** - Return a copy of the frequency.
- **QuarterBegin.is_anchored ()** - Return boolean whether the frequency is a unit frequency (n\=1).
- **QuarterBegin.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **QuarterBegin.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **QuarterBegin.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **QuarterBegin.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **QuarterBegin.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **QuarterBegin.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **QuarterBegin.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

BYearEnd

- **BYearEnd** - DateOffset increments between the last business day of the year.

- **BYearEnd.freqstr** - Return a string representing the frequency.

- **BYearEnd.kwds** - Return a dict of extra parameters for the offset.
- **BYearEnd.name** - Return a string representing the base frequency.
- **BYearEnd.nanos** -
- **BYearEnd.normalize** -
- **BYearEnd.rule_code** -
- **BYearEnd.n** -
- **BYearEnd.month** -
- **BYearEnd.copy ()** - Return a copy of the frequency.
- **BYearEnd.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **BYearEnd.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **BYearEnd.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **BYearEnd.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **BYearEnd.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **BYearEnd.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **BYearEnd.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **BYearEnd.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

BYearBegin

- **BYearBegin** - DateOffset increments between the first business day of the year.

- **BYearBegin.freqstr** - Return a string representing the frequency.
- **BYearBegin.kwds** - Return a dict of extra parameters for the offset.
- **BYearBegin.name** - Return a string representing the base frequency.
- **BYearBegin.nanos** -
- **BYearBegin.normalize** -
- **BYearBegin.rule_code** -
- **BYearBegin.n** -
- **BYearBegin.month** -
- **BYearBegin.copy ()** - Return a copy of the frequency.
- **BYearBegin.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **BYearBegin.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **BYearBegin.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **BYearBegin.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.

- **BYearBegin.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **BYearBegin.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **BYearBegin.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **BYearBegin.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

YearEnd

- **YearEnd** - DateOffset increments between calendar year end dates.

- **YearEnd.freqstr** - Return a string representing the frequency.
- **YearEnd.kwds** - Return a dict of extra parameters for the offset.
- **YearEnd.name** - Return a string representing the base frequency.
- **YearEnd.nanos** -
- **YearEnd.normalize** -
- **YearEnd.rule_code** -
- **YearEnd.n** -
- **YearEnd.month** -
- **YearEnd.copy ()** - Return a copy of the frequency.
- **YearEnd.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **YearEnd.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **YearEnd.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **YearEnd.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **YearEnd.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **YearEnd.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **YearEnd.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **YearEnd.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

YearBegin

- **YearBegin** - DateOffset increments between calendar year begin dates.

- **YearBegin.freqstr** - Return a string representing the frequency.

- **YearBegin.kwds** - Return a dict of extra parameters for the offset.
- **YearBegin.name** - Return a string representing the base frequency.
- **YearBegin.nanos** -
- **YearBegin.normalize** -
- **YearBegin.rule_code** -
- **YearBegin.n** -
- **YearBegin.month** -
- **YearBegin.copy ()** - Return a copy of the frequency.
- **YearBegin.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **YearBegin.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **YearBegin.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **YearBegin.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **YearBegin.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **YearBegin.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **YearBegin.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **YearBegin.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

FY5253

- **FY5253** - Describes 52-53 week fiscal year.

- **FY5253.freqstr** - Return a string representing the frequency.
- **FY5253.kwds** - Return a dict of extra parameters for the offset.
- **FY5253.name** - Return a string representing the base frequency.
- **FY5253.nanos** -
- **FY5253.normalize** -
- **FY5253.rule_code** -
- **FY5253.n** -
- **FY5253.startingMonth** -
- **FY5253.variation** -
- **FY5253.weekday** -
- **FY5253.copy ()** - Return a copy of the frequency.
- **FY5253.get_rule_code_suffix ()** -
- **FY5253.get_year_end (dt)** -
- **FY5253.is_anchored ()** - Return boolean whether the frequency is a unit frequency (n\=1).
- **FY5253.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **FY5253.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.

- **FY5253.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **FY5253.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **FY5253.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **FY5253.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **FY5253.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

FY5253Quarter

- **FY5253Quarter** - DateOffset increments between business quarter dates for 52-53 week fiscal year.

- **FY5253Quarter.freqstr** - Return a string representing the frequency.
- **FY5253Quarter.kwds** - Return a dict of extra parameters for the offset.
- **FY5253Quarter.name** - Return a string representing the base frequency.
- **FY5253Quarter.nanos** -
- **FY5253Quarter.normalize** -
- **FY5253Quarter.rule_code** -
- **FY5253Quarter.n** -
- **FY5253Quarter.qtr_with_extra_week** -
- **FY5253Quarter.startingMonth** -
- **FY5253Quarter.variation** -
- **FY5253Quarter.weekday** -
- **FY5253Quarter.copy ()** - Return a copy of the frequency.
- **FY5253Quarter.get_rule_code_suffix ()** -
- **FY5253Quarter.get_weeks (dt)** -
- **FY5253Quarter.is_anchored ()** - Return boolean whether the frequency is a unit frequency (n\=1).
- **FY5253Quarter.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **FY5253Quarter.year_has_extra_week (dt)** -
- **FY5253Quarter.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **FY5253Quarter.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **FY5253Quarter.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **FY5253Quarter.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **FY5253Quarter.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **FY5253Quarter.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Easter

- **Easter** - DateOffset for the Easter holiday using logic defined in dateutil.

- **Easter.freqstr** - Return a string representing the frequency.
- **Easter.kwds** - Return a dict of extra parameters for the offset.
- **Easter.name** - Return a string representing the base frequency.
- **Easter.nanos** -
- **Easter.normalize** -
- **Easter.rule_code** -
- **Easter.n** -
- **Easter.copy ()** - Return a copy of the frequency.
- **Easter.is_anchored ()** - (DEPRECATED) Return boolean whether the frequency is a unit frequency (n\=1).
- **Easter.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Easter.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Easter.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Easter.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Easter.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Easter.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Easter.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Tick

- **Tick** -

- **Tick.delta** -
- **Tick.freqstr** - Return a string representing the frequency.
- **Tick.kwds** - Return a dict of extra parameters for the offset.
- **Tick.name** - Return a string representing the base frequency.
- **Tick.nanos** - Return an integer of the total number of nanoseconds.
- **Tick.normalize** -
- **Tick.rule_code** -
- **Tick.n** -
- **Tick.copy ()** - Return a copy of the frequency.
- **Tick.is_anchored ()** - (DEPRECATED) Return False.
- **Tick.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Tick.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.

- **Tick.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Tick.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Tick.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Tick.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Tick.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Day

- **Day** - Offset n days.

- **Day.delta** -
- **Day.freqstr** - Return a string representing the frequency.
- **Day.kwds** - Return a dict of extra parameters for the offset.
- **Day.name** - Return a string representing the base frequency.
- **Day.nanos** - Return an integer of the total number of nanoseconds.
- **Day.normalize** -
- **Day.rule_code** -
- **Day.n** -
- **Day.copy ()** - Return a copy of the frequency.
- **Day.is_anchored ()** - (DEPRECATED) Return False.
- **Day.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Day.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Day.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Day.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Day.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Day.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Day.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Hour

- **Hour** - Offset n hours.

- **Hour.delta** -
- **Hour.freqstr** - Return a string representing the frequency.
- **Hour.kwds** - Return a dict of extra parameters for the offset.
- **Hour.name** - Return a string representing the base frequency.

- **Hour.nanos** - Return an integer of the total number of nanoseconds.
- **Hour.normalize** -
- **Hour.rule_code** -
- **Hour.n** -
- **Hour.copy ()** - Return a copy of the frequency.
- **Hour.is_anchored ()** - (DEPRECATED) Return False.
- **Hour.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Hour.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Hour.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Hour.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Hour.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Hour.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Hour.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Minute

- **Minute** - Offset n minutes.

- **Minute.delta** -
- **Minute.freqstr** - Return a string representing the frequency.
- **Minute.kwds** - Return a dict of extra parameters for the offset.
- **Minute.name** - Return a string representing the base frequency.
- **Minute.nanos** - Return an integer of the total number of nanoseconds.
- **Minute.normalize** -
- **Minute.rule_code** -
- **Minute.n** -
- **Minute.copy ()** - Return a copy of the frequency.
- **Minute.is_anchored ()** - (DEPRECATED) Return False.
- **Minute.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Minute.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Minute.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Minute.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Minute.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Minute.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Minute.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Second

- **Second** - Offset n seconds.

- **Second.delta** -
- **Second.freqstr** - Return a string representing the frequency.
- **Second.kwds** - Return a dict of extra parameters for the offset.
- **Second.name** - Return a string representing the base frequency.
- **Second.nanos** - Return an integer of the total number of nanoseconds.
- **Second.normalize** -
- **Second.rule_code** -
- **Second.n** -
- **Second.copy ()** - Return a copy of the frequency.
- **Second.is_anchored ()** - (DEPRECATED) Return False.
- **Second.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Second.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Second.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Second.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Second.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Second.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Second.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Milli

- **Milli** - Offset n milliseconds.

- **Milli.delta** -
- **Milli.freqstr** - Return a string representing the frequency.
- **Milli.kwds** - Return a dict of extra parameters for the offset.
- **Milli.name** - Return a string representing the base frequency.
- **Milli.nanos** - Return an integer of the total number of nanoseconds.
- **Milli.normalize** -
- **Milli.rule_code** -
- **Milli.n** -
- **Milli.copy ()** - Return a copy of the frequency.
- **Milli.is_anchored ()** - (DEPRECATED) Return False.

- **Milli.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Milli.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Milli.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Milli.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Milli.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Milli.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Milli.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Micro

- **Micro** - Offset n microseconds.

- **Micro.delta** -
- **Micro.freqstr** - Return a string representing the frequency.
- **Micro.kwds** - Return a dict of extra parameters for the offset.
- **Micro.name** - Return a string representing the base frequency.
- **Micro.nanos** - Return an integer of the total number of nanoseconds.
- **Micro.normalize** -
- **Micro.rule_code** -
- **Micro.n** -
- **Micro.copy ()** - Return a copy of the frequency.
- **Micro.is_anchored ()** - (DEPRECATED) Return False.
- **Micro.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Micro.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Micro.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Micro.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Micro.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Micro.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.

- **Micro.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Nano

- **Nano** - Offset n nanoseconds.

- **Nano.delta** -
- **Nano.freqstr** - Return a string representing the frequency.
- **Nano.kwds** - Return a dict of extra parameters for the offset.
- **Nano.name** - Return a string representing the base frequency.
- **Nano.nanos** - Return an integer of the total number of nanoseconds.
- **Nano.normalize** -
- **Nano.rule_code** -
- **Nano.n** -
- **Nano.copy ()** - Return a copy of the frequency.
- **Nano.is_anchored ()** - (DEPRECATED) Return False.
- **Nano.is_on_offset (dt)** - Return boolean whether a timestamp intersects with this frequency.
- **Nano.is_month_start (ts)** - Return boolean whether a timestamp occurs on the month start.
- **Nano.is_month_end (ts)** - Return boolean whether a timestamp occurs on the month end.
- **Nano.is_quarter_start (ts)** - Return boolean whether a timestamp occurs on the quarter start.
- **Nano.is_quarter_end (ts)** - Return boolean whether a timestamp occurs on the quarter end.
- **Nano.is_year_start (ts)** - Return boolean whether a timestamp occurs on the year start.
- **Nano.is_year_end (ts)** - Return boolean whether a timestamp occurs on the year end.

Window

Rolling window functions

- **Rolling.count ([numeric_only])** - Calculate the rolling count of non NaN observations.
- **Rolling.sum ([numeric_only, engine, ...])** - Calculate the rolling sum.
- **Rolling.mean ([numeric_only, engine, ...])** - Calculate the rolling mean.
- **Rolling.median ([numeric_only, engine, ...])** - Calculate the rolling median.
- **Rolling.var ([ddof, numeric_only, engine, ...])** - Calculate the rolling variance.
- **Rolling.std ([ddof, numeric_only, engine, ...])** - Calculate the rolling standard deviation.
- **Rolling.min ([numeric_only, engine, ...])** - Calculate the rolling minimum.
- **Rolling.max ([numeric_only, engine, ...])** - Calculate the rolling maximum.
- **Rolling.corr ([other, pairwise, ddof, ...])** - Calculate the rolling correlation.
- **Rolling.cov ([other, pairwise, ddof, ...])** - Calculate the rolling sample covariance.

- **Rolling.skew ([numeric_only])** - Calculate the rolling unbiased skewness.
- **Rolling.kurt ([numeric_only])** - Calculate the rolling Fisher's definition of kurtosis without bias.
- **Rolling.apply (func[, raw, engine, ...])** - Calculate the rolling custom aggregation function.
- **Rolling.aggregate (func, *args, **kwargs)** - Aggregate using one or more operations over the specified axis.
- **Rolling.quantile (q[, interpolation, ...])** - Calculate the rolling quantile.
- **Rolling.sem ([ddof, numeric_only])** - Calculate the rolling standard error of mean.

- **Rolling.rank ([method, ascending, pct, ...])** - Calculate the rolling rank.

Weighted window functions

- **Window.mean ([numeric_only])** - Calculate the rolling weighted window mean.

- **Window.sum ([numeric_only])** - Calculate the rolling weighted window sum.
- **Window.var ([ddof, numeric_only])** - Calculate the rolling weighted window variance.

- **Window.std ([ddof, numeric_only])** - Calculate the rolling weighted window standard deviation.

Expanding window functions

- **Expanding.count ([numeric_only])** - Calculate the expanding count of non NaN observations.

- **Expanding.sum ([numeric_only, engine, ...])** - Calculate the expanding sum.
- **Expanding.mean ([numeric_only, engine, ...])** - Calculate the expanding mean.
- **Expanding.median ([numeric_only, engine, ...])** - Calculate the expanding median.
- **Expanding.var ([ddof, numeric_only, engine, ...])** - Calculate the expanding variance.
- **Expanding.std ([ddof, numeric_only, engine, ...])** - Calculate the expanding standard deviation.
- **Expanding.min ([numeric_only, engine, ...])** - Calculate the expanding minimum.
- **Expanding.max ([numeric_only, engine, ...])** - Calculate the expanding maximum.
- **Expanding.corr ([other, pairwise, ddof, ...])** - Calculate the expanding correlation.
- **Expanding.cov ([other, pairwise, ddof, ...])** - Calculate the expanding sample covariance.

- **Expanding.skew ([numeric_only])** - Calculate the expanding unbiased skewness.
- **Expanding.kurt ([numeric_only])** - Calculate the expanding Fisher's definition of kurtosis without bias.
- **Expanding.apply (func[, raw, engine, ...])** - Calculate the expanding custom aggregation function.
- **Expanding.aggregate (func, *args, **kwargs)** - Aggregate using one or more operations over the specified axis.
- **Expanding.quantile (q[, interpolation, ...])** - Calculate the expanding quantile.
- **Expanding.sem ([ddof, numeric_only])** - Calculate the expanding standard error of mean.

- **Expanding.rank ([method, ascending, pct, ...])** - Calculate the expanding rank.

Exponentially-weighted window functions

- **ExponentialMovingWindow.mean ([numeric_only, ...])** - Calculate the ewm (exponential weighted moment) mean.

- **ExponentialMovingWindow.sum ([numeric_only, ...])** - Calculate the ewm (exponential weighted moment) sum.
- **ExponentialMovingWindow.std ([bias, numeric_only])** - Calculate the ewm (exponential weighted moment) standard deviation.
- **ExponentialMovingWindow.var ([bias, numeric_only])** - Calculate the ewm (exponential weighted moment) variance.
- **ExponentialMovingWindow.corr ([other, ...])** - Calculate the ewm (exponential weighted moment) sample correlation.

- **ExponentialMovingWindow.cov ([other, ...])** - Calculate the ewm (exponential weighted moment) sample covariance.

Window indexer

- **api.indexers.BaseIndexer ([index_array, ...])** - Base class for window bounds calculations.

- **api.indexers.FixedForwardWindowIndexer ([...])** - Creates window boundaries for fixed-length windows that include the current row.
- **api.indexers.VariableOffsetWindowIndexer ([...])** - Calculate window boundaries based on a non-fixed offset such as a BusinessDay.

GroupBy

Indexing, iteration

- **DataFrameGroupBy.__iter__ ()** - Groupby iterator.
- **SeriesGroupBy.__iter__ ()** - Groupby iterator.
- **DataFrameGroupBy.groups** - Dict {group name -> group labels}.
- **SeriesGroupBy.groups** - Dict {group name -> group labels}.
- **DataFrameGroupBy.indices** - Dict {group name -> group indices}.
- **SeriesGroupBy.indices** - Dict {group name -> group indices}.
- **DataFrameGroupBy.get_group (name[, obj])** - Construct DataFrame from group with provided name.
- **SeriesGroupBy.get_group (name[, obj])** - Construct DataFrame from group with provided name.

- **Grouper (*args, **kwargs)** - A Grouper allows the user to specify a groupby instruction for an object.

Function application helper

- **NamedAgg (column, aggfunc)** - Helper for column specific aggregation with control over output column names.

Function application

- **SeriesGroupBy.apply (func, *args, **kwargs)** - Apply function func group-wise and combine the results together.

- **DataFrameGroupBy.apply (func, *args[, ...])** - Apply function func group-wise and combine the results together.
- **SeriesGroupBy.agg ([func, engine, engine_kwargs])** - Aggregate using one or more operations over the specified axis.
- **DataFrameGroupBy.agg ([func, engine, ...])** - Aggregate using one or more operations over the specified axis.
- **SeriesGroupBy.aggregate ([func, engine, ...])** - Aggregate using one or more operations over the specified axis.
- **DataFrameGroupBy.aggregate ([func, engine, ...])** - Aggregate using one or more operations over the specified axis.
- **SeriesGroupBy.transform (func, *args[, ...])** - Call function producing a same-indexed Series on each group.
- **DataFrameGroupBy.transform (func, *args[, ...])** - Call function producing a same-indexed DataFrame on each group.
- **SeriesGroupBy.pipe (func, *args, **kwargs)** - Apply a func with arguments to this GroupBy object and return its result.
- **DataFrameGroupBy.pipe (func, *args, **kwargs)** - Apply a func with arguments to this GroupBy object and return its result.
- **DataFrameGroupBy.filter (func[, dropna])** - Filter elements from groups that don't satisfy a criterion.

- **SeriesGroupBy.filter (func[, dropna])** - Filter elements from groups that don't satisfy a criterion.

DataFrameGroupBy computations / descriptive stats

- **DataFrameGroupBy.all ([skipna])** - Return True if all values in the group are truthful, else False.

- **DataFrameGroupBy.any ([skipna])** - Return True if any value in the group is truthful, else False.
- **DataFrameGroupBy.bfill ([limit])** - Backward fill the values.
- **DataFrameGroupBy.corr ([method, min_periods, ...])** - Compute pairwise correlation of columns, excluding NA/null values.
- **DataFrameGroupBy.corrwith (other[, axis, ...])** - Compute pairwise correlation.
- **DataFrameGroupBy.count ()** - Compute count of group, excluding missing values.
- **DataFrameGroupBy.cov ([min_periods, ddof, ...])** - Compute pairwise covariance of columns, excluding NA/null values.
- **DataFrameGroupBy.cumcount ([ascending])** - Number each item in each group from 0 to the length of that group - 1.
- **DataFrameGroupBy.cummax ([axis, numeric_only])** - Cumulative max for each group.
- **DataFrameGroupBy.cummin ([axis, numeric_only])** - Cumulative min for each group.
- **DataFrameGroupBy.cumprod ([axis])** - Cumulative product for each group.
- **DataFrameGroupBy.cumsum ([axis])** - Cumulative sum for each group.
- **DataFrameGroupBy.describe ([percentiles, ...])** - Generate descriptive statistics.
- **DataFrameGroupBy.diff ([periods, axis])** - First discrete difference of element.
- **DataFrameGroupBy.ffill ([limit])** - Forward fill the values.
- **DataFrameGroupBy.fillna ([value, method, ...])** - (DEPRECATED) Fill NA/NaN values using the specified method within groups.
- **DataFrameGroupBy.first ([numeric_only, ...])** - Compute the first entry of each column within each group.
- **DataFrameGroupBy.head ([n])** - Return first n rows of each group.
- **DataFrameGroupBy.idxmax ([axis, skipna, ...])** - Return index of first occurrence of maximum over requested axis.
- **DataFrameGroupBy.idxmin ([axis, skipna, ...])** - Return index of first occurrence of minimum over requested axis.
- **DataFrameGroupBy.last ([numeric_only, ...])** - Compute the last entry of each column within each group.
- **DataFrameGroupBy.max ([numeric_only, ...])** - Compute max of group values.
- **DataFrameGroupBy.mean ([numeric_only, ...])** - Compute mean of groups, excluding missing values.

- **DataFrameGroupBy.median ([numeric_only])** - Compute median of groups, excluding missing values.
- **DataFrameGroupBy.min ([numeric_only, ...])** - Compute min of group values.
- **DataFrameGroupBy.ngroup ([ascending])** - Number each group from 0 to the number of groups - 1.
- **DataFrameGroupBy.nth** - Take the nth row from each group if n is an int, otherwise a subset of rows.
- **DataFrameGroupBy.nunique ([dropna])** - Return DataFrame with counts of unique elements in each position.
- **DataFrameGroupBy.ohlc ()** - Compute open, high, low and close values of a group, excluding missing values.
- **DataFrameGroupBy.pct_change ([periods, ...])** - Calculate pct_change of each value to previous entry in group.
- **DataFrameGroupBy.prod ([numeric_only, min_count])** - Compute prod of group values.
- **DataFrameGroupBy.quantile ([q, ...])** - Return group values at the given quantile, a la numpy.percentile.
- **DataFrameGroupBy.rank ([method, ascending, ...])** - Provide the rank of values within each group.
- **DataFrameGroupBy.resample (rule, *args[, ...])** - Provide resampling when using a TimeGrouper.
- **DataFrameGroupBy.rolling (*args, **kwargs)** - Return a rolling grouper, providing rolling functionality per group.
- **DataFrameGroupBy.sample ([n, frac, replace, ...])** - Return a random sample of items from each group.
- **DataFrameGroupBy.sem ([ddof, numeric_only])** - Compute standard error of the mean of groups, excluding missing values.
- **DataFrameGroupBy.shift ([periods, freq, ...])** - Shift each group by periods observations.
- **DataFrameGroupBy.size ()** - Compute group sizes.
- **DataFrameGroupBy.skew ([axis, skipna, ...])** - Return unbiased skew within groups.
- **DataFrameGroupBy.std ([ddof, engine, ...])** - Compute standard deviation of groups, excluding missing values.
- **DataFrameGroupBy.sum ([numeric_only, ...])** - Compute sum of group values.
- **DataFrameGroupBy.var ([ddof, engine, ...])** - Compute variance of groups, excluding missing values.
- **DataFrameGroupBy.tail ([n])** - Return last n rows of each group.
- **DataFrameGroupBy.take (indices[, axis])** - Return the elements in the given positional indices in each group.

- **DataFrameGroupBy.value_counts ([subset, ...])** - Return a Series or DataFrame containing counts of unique rows.

SeriesGroupBy computations / descriptive stats

- **SeriesGroupBy.all ([skipna])** - Return True if all values in the group are truthful, else False.

- **SeriesGroupBy.any ([skipna])** - Return True if any value in the group is truthful, else False.
- **SeriesGroupBy.bfill ([limit])** - Backward fill the values.
- **SeriesGroupBy.corr (other[, method, min_periods])** - Compute correlation with other Series, excluding missing values.
- **SeriesGroupBy.count ()** - Compute count of group, excluding missing values.
- **SeriesGroupBy.cov (other[, min_periods, ddof])** - Compute covariance with Series, excluding missing values.
- **SeriesGroupBy.cumcount ([ascending])** - Number each item in each group from 0 to the length of that group - 1.
- **SeriesGroupBy.cummax ([axis, numeric_only])** - Cumulative max for each group.
- **SeriesGroupBy.cummin ([axis, numeric_only])** - Cumulative min for each group.
- **SeriesGroupBy.cumprod ([axis])** - Cumulative product for each group.
- **SeriesGroupBy.cumsum ([axis])** - Cumulative sum for each group.
- **SeriesGroupBy.describe ([percentiles, ...])** - Generate descriptive statistics.
- **SeriesGroupBy.diff ([periods, axis])** - First discrete difference of element.
- **SeriesGroupBy.ffill ([limit])** - Forward fill the values.
- **SeriesGroupBy.fillna ([value, method, axis, ...])** - (DEPRECATED) Fill NA/NaN values using the specified method within groups.
- **SeriesGroupBy.first ([numeric_only, ...])** - Compute the first entry of each column within each group.
- **SeriesGroupBy.head ([n])** - Return first n rows of each group.
- **SeriesGroupBy.last ([numeric_only, ...])** - Compute the last entry of each column within each group.
- **SeriesGroupBy.idxmax ([axis, skipna])** - Return the row label of the maximum value.
- **SeriesGroupBy.idxmin ([axis, skipna])** - Return the row label of the minimum value.
- **SeriesGroupBy.is_monotonic_increasing** - Return whether each group's values are monotonically increasing.
- **SeriesGroupBy.is_monotonic_decreasing** - Return whether each group's values are monotonically decreasing.
- **SeriesGroupBy.max ([numeric_only, min_count, ...])** - Compute max of group values.
- **SeriesGroupBy.mean ([numeric_only, engine, ...])** - Compute mean of groups, excluding missing values.
- **SeriesGroupBy.median ([numeric_only])** - Compute median of groups, excluding missing values.

- **SeriesGroupBy.min ([numeric_only, min_count, ...])** - Compute min of group values.
- **SeriesGroupBy.ngroup ([ascending])** - Number each group from 0 to the number of groups - 1.
- **SeriesGroupBy.nlargest ([n, keep])** - Return the largest n elements.
- **SeriesGroupBy.nsmallest ([n, keep])** - Return the smallest n elements.
- **SeriesGroupBy.nth** - Take the nth row from each group if n is an int, otherwise a subset of rows.
- **SeriesGroupBy.nunique ([dropna])** - Return number of unique elements in the group.
- **SeriesGroupBy.unique ()** - Return unique values for each group.
- **SeriesGroupBy.ohlc ()** - Compute open, high, low and close values of a group, excluding missing values.
- **SeriesGroupBy.pct_change ([periods, ...])** - Calculate pct_change of each value to previous entry in group.
- **SeriesGroupBy.prod ([numeric_only, min_count])** - Compute prod of group values.
- **SeriesGroupBy.quantile ([q, interpolation, ...])** - Return group values at the given quantile, a la numpy.percentile.
- **SeriesGroupBy.rank ([method, ascending, ...])** - Provide the rank of values within each group.
- **SeriesGroupBy.resample (rule, *args[, ...])** - Provide resampling when using a TimeGrouper.
- **SeriesGroupBy.rolling (*args, **kwargs)** - Return a rolling grouper, providing rolling functionality per group.
- **SeriesGroupBy.sample ([n, frac, replace, ...])** - Return a random sample of items from each group.
- **SeriesGroupBy.sem ([ddof, numeric_only])** - Compute standard error of the mean of groups, excluding missing values.
- **SeriesGroupBy.shift ([periods, freq, axis, ...])** - Shift each group by periods observations.
- **SeriesGroupBy.size ()** - Compute group sizes.
- **SeriesGroupBy.skew ([axis, skipna, numeric_only])** - Return unbiased skew within groups.
- **SeriesGroupBy.std ([ddof, engine, ...])** - Compute standard deviation of groups, excluding missing values.
- **SeriesGroupBy.sum ([numeric_only, min_count, ...])** - Compute sum of group values.
- **SeriesGroupBy.var ([ddof, engine, ...])** - Compute variance of groups, excluding missing values.
- **SeriesGroupBy.tail ([n])** - Return last n rows of each group.
- **SeriesGroupBy.take (indices[, axis])** - Return the elements in the given positional indices in each group.

- **SeriesGroupBy.value_counts ([normalize, ...])** -

Plotting and visualization

- **DataFrameGroupBy.boxplot ([subplots, column, ...])** - Make box plots from DataFrameGroupBy data.

- **DataFrameGroupBy.hist ([column, by, grid, ...])** - Make a histogram of the DataFrame's columns.
- **SeriesGroupBy.hist ([by, ax, grid, ...])** - Draw histogram of the input series using matplotlib.
- **DataFrameGroupBy.plot** - Make plots of Series or DataFrame.
- **SeriesGroupBy.plot** - Make plots of Series or DataFrame.

Resampling

Indexing, iteration

- **Resampler.__iter__ ()** - Groupby iterator.
- **Resampler.groups** - Dict {group name -> group labels}.
- **Resampler.indices** - Dict {group name -> group indices}.

- **Resampler.get_group (name[, obj])** - Construct DataFrame from group with provided name.

Function application

- **Resampler.apply ([func])** - Aggregate using one or more operations over the specified axis.

- **Resampler.aggregate ([func])** - Aggregate using one or more operations over the specified axis.
- **Resampler.transform (arg, *args, **kwargs)** - Call function producing a like-indexed Series on each group.

- **Resampler.pipe (func, *args, **kwargs)** - Apply a func with arguments to this Resampler object and return its result.

Upsampling

- **Resampler.ffill ([limit])** - Forward fill the values.

- **Resampler.bfill ([limit])** - Backward fill the new missing values in the resampled data.
- **Resampler.nearest ([limit])** - Resample by using the nearest value.
- **Resampler.fillna (method[, limit])** - Fill missing values introduced by upsampling.
- **Resampler.asfreq ([fill_value])** - Return the values at the new freq, essentially a reindex.

- **Resampler.interpolate ([method, axis, limit, ...])** - Interpolate values between target timestamps according to different methods.

Computations / descriptive stats

- **Resampler.count ()** - Compute count of group, excluding missing values.
- **Resampler.nunique (*args, **kwargs)** - Return number of unique elements in the group.
- **Resampler.first ([numeric_only, min_count, ...])** - Compute the first entry of each column within each group.
- **Resampler.last ([numeric_only, min_count, skipna])** - Compute the last entry of each column within each group.
- **Resampler.max ([numeric_only, min_count])** - Compute max value of group.
- **Resampler.mean ([numeric_only])** - Compute mean of groups, excluding missing values.
- **Resampler.median ([numeric_only])** - Compute median of groups, excluding missing values.
- **Resampler.min ([numeric_only, min_count])** - Compute min value of group.
- **Resampler.ohlc (*args, **kwargs)** - Compute open, high, low and close values of a group, excluding missing values.
- **Resampler.prod ([numeric_only, min_count])** - Compute prod of group values.
- **Resampler.size ()** - Compute group sizes.
- **Resampler.sem ([ddof, numeric_only])** - Compute standard error of the mean of groups, excluding missing values.
- **Resampler.std ([ddof, numeric_only])** - Compute standard deviation of groups, excluding missing values.
- **Resampler.sum ([numeric_only, min_count])** - Compute sum of group values.
- **Resampler.var ([ddof, numeric_only])** - Compute variance of groups, excluding missing values.
- **Resampler.quantile ([q])** - Return value at the given quantile.

Style

Styler constructor

- **Styler (data[, precision, table_styles, ...])** - Helps style a DataFrame or Series according to the data with HTML and CSS.
- **Styler.from_custom_template (searchpath[, ...])** - Factory function for creating a subclass of Styler .

Styler properties

- **Styler.env** -
- **Styler.template_html** -

- **Styler.template_html_style** -
- **Styler.template_html_table** -
- **Styler.template_latex** -
- **Styler.template_string** -

- **Styler.loader** -

Style application

- **Styler.apply (func[, axis, subset])** - Apply a CSS-styling function column-wise, row-wise, or table-wise.

- **Styler.map (func[, subset])** - Apply a CSS-styling function elementwise.
- **Styler.apply_index (func[, axis, level])** - Apply a CSS-styling function to the index or column headers, level-wise.
- **Styler.map_index (func[, axis, level])** - Apply a CSS-styling function to the index or column headers, elementwise.
- **Styler.format ([formatter, subset, na_rep, ...])** - Format the text display value of cells.
- **Styler.format_index ([formatter, axis, ...])** - Format the text display value of index labels or column headers.
- **Styler.relabel_index (labels[, axis, level])** - Relabel the index, or column header, keys to display a set of specified values.
- **Styler.hide ([subset, axis, level, names])** - Hide the entire index / column headers, or specific rows / columns from display.
- **Styler.concat (other)** - Append another Styler to combine the output into a single table.
- **Styler.set_td_classes (classes)** - Set the class attribute of
- **Styler.set_table_styles ([table_styles, ...])** - Set the table styles included within the
- **Styler.set_table_attributes (attributes)** - Set the table attributes added to the
- **Styler.set_tooltips (ttips[, props, css_class])** - Set the DataFrame of strings on Styler generating :hover tooltips.
- **Styler.set_caption (caption)** - Set the text added to a
- **Styler.set_sticky ([axis, pixel_size, levels])** - Add CSS to permanently display the index or column headers in a scrolling frame.
- **Styler.set_properties ([subset])** - Set defined CSS-properties to each
- **Styler.set_uuid (uuid)** - Set the uuid applied to id attributes of HTML elements.
- **Styler.clear ()** - Reset the Styler , removing any previously applied styles.

- **Styler.pipe (func, *args, **kwargs)** - Apply func(self, *args, **kwargs) , and return the result.

Builtin styles

- **Styler.highlight_null ([color, subset, props])** - Highlight missing values with a style.

- **Styler.highlight_max ([subset, color, axis, ...])** - Highlight the maximum with a style.
- **Styler.highlight_min ([subset, color, axis, ...])** - Highlight the minimum with a style.
- **Styler.highlight_between ([subset, color, ...])** - Highlight a defined range with a style.
- **Styler.highlight_quantile ([subset, color, ...])** - Highlight values defined by a quantile with a style.
- **Styler.background_gradient ([cmap, low, ...])** - Color the background in a gradient style.
- **Styler.text_gradient ([cmap, low, high, ...])** - Color the text in a gradient style.

- **Styler.bar ([subset, axis, color, cmap, ...])** - Draw bar chart in the cell backgrounds.

Style export and import

- **Styler.to_html ([buf, table_uuid, ...])** - Write Styler to a file, buffer or string in HTML-CSS format.

- **Styler.to_latex ([buf, column_format, ...])** - Write Styler to a file, buffer or string in LaTeX format.
- **Styler.to_excel (excel_writer[, sheet_name, ...])** - Write Styler to an Excel sheet.
- **Styler.to_string ([buf, encoding, ...])** - Write Styler to a file, buffer or string in text format.
- **Styler.export ()** - Export the styles applied to the current Styler.
- **Styler.use (styles)** - Set the styles on the current Styler.

Plotting

- **andrews_curves (frame, class_column[, ax, ...])** - Generate a matplotlib plot for visualizing clusters of multivariate data.
- **autocorrelation_plot (series[, ax])** - Autocorrelation plot for time series.
- **bootstrap_plot (series[, fig, size, samples])** - Bootstrap plot on mean, median and mid-range statistics.
- **boxplot (data[, column, by, ax, fontsize, ...])** - Make a box plot from DataFrame columns.
- **deregister_matplotlib_converters ()** - Remove pandas formatters and converters.
- **lag_plot (series[, lag, ax])** - Lag plot for time series.

- **parallel_coordinates (frame, class_column[, ...])** - Parallel coordinates plotting.
- **plot_params** - Stores pandas plotting options.
- **radviz (frame, class_column[, ax, color, ...])** - Plot a multidimensional dataset in 2D.
- **register_matplotlib_converters ()** - Register pandas formatters and converters with matplotlib.
- **scatter_matrix (frame[, alpha, figsize, ax, ...])** - Draw a matrix of scatter plots.
- **table (ax, data, **kwargs)** - Helper function to convert DataFrame and Series to matplotlib.table.

Options and settings

Working with options

- **describe_option (pat[, _print_desc])** - Prints the description for one or more registered options.
- **reset_option (pat)** - Reset one or more options to their default value.
- **get_option (pat)** - Retrieves the value of the specified option.
- **set_option (pat, value)** - Sets the value of the specified option.

- **option_context (*args)** - Context manager to temporarily set options in the with statement context.

Numeric formatting

- **set_eng_float_format ([accuracy, use_eng_prefix])** - Format float representation in DataFrame with SI notation.

Extensions

- **api.extensions.register_extension_dtype (cls)** - Register an ExtensionType with pandas as class decorator.
- **api.extensions.register_dataframe_accessor (name)** - Register a custom accessor on DataFrame objects.
- **api.extensions.register_series_accessor (name)** - Register a custom accessor on Series objects.
- **api.extensions.register_index_accessor (name)** - Register a custom accessor on Index objects.
- **api.extensions.ExtensionDtype ()** - A custom data type, to be paired with an ExtensionArray.
- **api.extensions.ExtensionArray ()** - Abstract base class for custom 1-D array types.
- **arrays.NumpyExtensionArray (values[, copy])** - A pandas ExtensionArray for NumPy data.

- **api.indexers.check_array_indexer (array, indexer)** - Check if indexer is a valid array indexer for array .

Testing

Assertion functions

- **testing.assert_frame_equal (left, right[, ...])** - Check that left and right DataFrame are equal.
- **testing.assert_series_equal (left, right[, ...])** - Check that left and right Series are equal.
- **testing.assert_index_equal (left, right[, ...])** - Check that left and right Index are equal.

- **testing.assert_extension_array_equal (left, right)** - Check that left and right ExtensionArrays are equal.

Exceptions and warnings

- **errors.AbstractMethodError (class_instance[, ...])** - Raise this error instead of NotImplementedError for abstract methods.

- **errors.AttributeConflictWarning** - Warning raised when index attributes conflict when using HDFStore.
- **errors.CategoricalConversionWarning** - Warning is raised when reading a partial labeled Stata file using a iterator.
- **errors.ChainedAssignmentError** - Warning raised when trying to set using chained assignment.
- **errors.ClosedFileError** - Exception is raised when trying to perform an operation on a closed HDFStore file.
- **errors.CSSWarning** - Warning is raised when converting css styling fails.
- **errors.DatabaseError** - Error is raised when executing sql with bad syntax or sql that throws an error.
- **errors.DataError** - Exceptionn raised when performing an operation on non-numerical data.
- **errors.DtypeWarning** - Warning raised when reading different dtypes in a column from a file.
- **errors.DuplicateLabelError** - Error raised when an operation would introduce duplicate labels.
- **errors.EmptyDataError** - Exception raised in pd.read_csv when empty data or header is encountered.
- **errors.IncompatibilityWarning** - Warning raised when trying to use where criteria on an incompatible HDF5 file.
- **errors.IndexingError** - Exception is raised when trying to index and there is a mismatch in dimensions.
- **errors.InvalidColumnName -** Warning raised by to_stata the column contains a non-valid stata name.
- **errors.InvalidComparison** - Exception is raised by _validate_comparison_value to indicate an invalid comparison.

- **errors.InvalidIndexError** - Exception raised when attempting to use an invalid index key.
- **errors.InvalidVersion** - An invalid version was found, users should refer to PEP 440.
- **errors.IntCastingNaNError** - Exception raised when converting (astype) an array with NaN to an integer type.
- **errors.LossySetitemError** - Raised when trying to do a __setitem__ on an np.ndarray that is not lossless.
- **errors.MergeError** - Exception raised when merging data.
- **errors.NoBufferPresent** - Exception is raised in _get_data_buffer to signal that there is no requested buffer.
- **errors.NullFrequencyError** - Exception raised when a freq cannot be null.
- **errors.NumbaUtilError** - Error raised for unsupported Numba engine routines.
- **errors.NumExprClobberingError** - Exception raised when trying to use a built-in numexpr name as a variable name.
- **errors.OptionError** - Exception raised for pandas.options.
- **errors.OutOfBoundsDatetime** - Raised when the datetime is outside the range that can be represented.
- **errors.OutOfBoundsTimedelta** - Raised when encountering a timedelta value that cannot be represented.
- **errors.ParserError** - Exception that is raised by an error encountered in parsing file contents.
- **errors.ParserWarning** - Warning raised when reading a file that doesn't use the default 'c' parser.
- **errors.PerformanceWarning** - Warning raised when there is a possible performance impact.
- **errors.PossibleDataLossError** - Exception raised when trying to open a HDFStore file when already opened.
- **errors.PossiblePrecisionLoss** - Warning raised by to_stata on a column with a value outside or equal to int64.
- **errors.PyperclipException** - Exception raised when clipboard functionality is unsupported.
- **errors.PyperclipWindowsException (message)** - Exception raised when clipboard functionality is unsupported by Windows.
- **errors.SettingWithCopyError** - Exception raised when trying to set on a copied slice from a DataFrame .
- **errors.SettingWithCopyWarning** - Warning raised when trying to set on a copied slice from a DataFrame .
- **errors.SpecificationError** - Exception raised by agg when the functions are ill-specified.
- **errors.UndefinedVariableError (name[, is_local])** - Exception raised by query or eval when using an undefined variable name.
- **errors.UnsortedIndexError** - Error raised when slicing a MultiIndex which has not been lexsorted.
- **errors.UnsupportedFunctionCall** - Exception raised when attempting to call a unsupported numpy function.

- **errors.ValueLabelTypeMismatch** - Warning raised by to_stata on a category column that contains non-string values.

Bug report function

- **show_versions ([as_json])** - Provide useful information, important for bug reports.

Test suite runner

- **test ([extra_args, run_doctests])** - Run the pandas test suite using pytest.

Missing values

- **NA** - NA ("not available") missing value indicator.
- **NaT** - (N)ot-(A)-(T)ime, the time equivalent of NaN.