Data Frames - Basic Operations

Here are some of the basic operations we typically perform on top of Pandas Data Frame.

- Getting number of records and columns.
- Getting data types of the columns.
- Replacing `NaN` with some standard values.
- Dropping a column from the Data Frame.
- Getting or updating column names.
- Sorting by index or values.

```
import pandas as pd
```

Note

Creating Pandas Data Frame using list of dicts.

```
sals_ld = [
    {'id': 1, 'sal': 1500.0},
    {'id': 2, 'sal': 2000.0, 'comm': 10.0},
    {'id': 3, 'sal': 2200.0, 'active': False}
]
```

Note

Column names will be inherited automatically using keys from the dict.

```
sals_df = pd.DataFrame(sals_ld)
```

```
sals_df
```

	id	sal	comm	active
0	1	1500.0	NaN	NaN
1	2	2000.0	10.0	NaN
2	3	2200.0	NaN	False

```
sals_df['id']
```

```
0    1
1    2
2    3
Name: id, dtype: int64
```

```
sals_df[['id', 'sal']]
```

	id	sal
0	1	1500.0
1	2	2000.0
2	3	2200.0

```
sals_df.shape
```

```
(3, 4)
```

```
sals_df.shape[0]
```

```
3
```

```
sals_df.count()
```

```
id        3
sal       3
comm      1
active    1
dtype: int64
```

```
sals_df.count()[:2]
```

```
id     3
sal    3
dtype: int64
```

```
sals_df.count()['id']
```

```
3
```

```
sals_df
```

	id	sal	comm	active
0	1	1500.0	NaN	NaN
1	2	2000.0	10.0	NaN
2	3	2200.0	NaN	False

```
sals_df.dtypes
```

```
id          int64
sal       float64
comm      float64
active     object
dtype: object
```

```
sals_df.fillna?
```

```
Signature:
sals_df.fillna(
    value=None,
    method=None,
    axis=None,
    inplace=False,
    limit=None,
    downcast=None,
) -> Union[_ForwardRef('DataFrame'), NoneType]
Docstring:
Fill NA/NaN values using the specified method.

Parameters
----------
value : scalar, dict, Series, or DataFrame
    Value to use to fill holes (e.g. 0), alternately a
    dict/Series/DataFrame of values specifying which value to use for
    each index (for a Series) or column (for a DataFrame).  Values not
    in the dict/Series/DataFrame will not be filled. This value cannot
    be a list.
method : {'backfill', 'bfill', 'pad', 'ffill', None}, default None
    Method to use for filling holes in reindexed Series
    pad / ffill: propagate last valid observation forward to next valid
    backfill / bfill: use next valid observation to fill gap.
axis : {0 or 'index', 1 or 'columns'}
    Axis along which to fill missing values.
inplace : bool, default False
    If True, fill in-place. Note: this will modify any
    other views on this object (e.g., a no-copy slice for a column in a
    DataFrame).
limit : int, default None
    If method is specified, this is the maximum number of consecutive
    NaN values to forward/backward fill. In other words, if there is
    a gap with more than this number of consecutive NaNs, it will only
    be partially filled. If method is not specified, this is the
    maximum number of entries along the entire axis where NaNs will be
    filled. Must be greater than 0 if not None.
downcast : dict, default is None
    A dict of item->dtype of what to downcast if possible,
    or the string 'infer' which will try to downcast to an appropriate
    equal type (e.g. float64 to int64 if possible).

Returns
-------
DataFrame or None
    Object with missing values filled or None if ``inplace=True``.

See Also
--------
interpolate : Fill NaN values using interpolation.
reindex : Conform object to new index.
asfreq : Convert TimeSeries to specified frequency.

Examples
--------
>>> df = pd.DataFrame([[np.nan, 2, np.nan, 0],
...                    [3, 4, np.nan, 1],
...                    [np.nan, np.nan, np.nan, 5],
...                    [np.nan, 3, np.nan, 4]],
...                   columns=list('ABCD'))
>>> df
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
3  NaN  3.0 NaN  4

Replace all NaN elements with 0s.

>>> df.fillna(0)
    A   B   C   D
0   0.0 2.0 0.0 0
1   3.0 4.0 0.0 1
2   0.0 0.0 0.0 5
3   0.0 3.0 0.0 4

We can also propagate non-null values forward or backward.

>>> df.fillna(method='ffill')
    A   B   C   D
0   NaN 2.0 NaN 0
1   3.0 4.0 NaN 1
2   3.0 4.0 NaN 5
3   3.0 3.0 NaN 4

Replace all NaN elements in column 'A', 'B', 'C', and 'D', with 0, 1,
2, and 3 respectively.
```

```
>>> values = {'A': 0, 'B': 1, 'C': 2, 'D': 3}
>>> df.fillna(value=values)
    A   B   C   D
0   0.0 2.0 2.0 0
1   3.0 4.0 2.0 1
2   0.0 1.0 2.0 5
3   0.0 3.0 2.0 4

Only replace the first NaN element.

>>> df.fillna(value=values, limit=1)
    A   B   C   D
0   0.0 2.0 2.0 0
1   3.0 4.0 NaN 1
2   NaN 1.0 NaN 5
3   NaN 3.0 NaN 4
File:      /opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/pandas/core/frame.py
Type:      method
```

```
sals_df.fillna(0.0)
```

	id	sal	comm	active
0	1	1500.0	0.0	0
1	2	2000.0	10.0	0
2	3	2200.0	0.0	False

```
sals_df.fillna({'comm': 0.0})
```

	id	sal	comm	active
0	1	1500.0	0.0	NaN
1	2	2000.0	10.0	NaN
2	3	2200.0	0.0	False

```
sals_df.fillna({'comm': 0.0, 'active': True})
```

	id	sal	comm	active
0	1	1500.0	0.0	True
1	2	2000.0	10.0	True
2	3	2200.0	0.0	False

Note

Original Data Frame will be untouched, instead a new Data Frame will be created. Original Data Frame still contain `NaN`. We typically assign the output of most of the Data Frame functions to another variable or object.

```
sals_df
```

	id	sal	comm	active
0	1	1500.0	NaN	NaN
1	2	2000.0	10.0	NaN
2	3	2200.0	NaN	False

```
sals_df = sals_df.fillna({'comm': 0.0, 'active': True})
sals_df
```

	id	sal	comm	active
0	1	1500.0	0.0	True
1	2	2000.0	10.0	True
2	3	2200.0	0.0	False

```
sals_df.drop?
```

```
Signature:
sals_df.drop(
    labels=None,
    axis=0,
    index=None,
    columns=None,
    level=None,
    inplace=False,
    errors='raise',
)
Docstring:
Drop specified labels from rows or columns.

Remove rows or columns by specifying label names and corresponding
axis, or by specifying directly index or column names. When using a
multi-index, labels on different levels can be removed by specifying
the level.

Parameters
----------
labels : single label or list-like
    Index or column labels to drop.
axis : {0 or 'index', 1 or 'columns'}, default 0
    Whether to drop labels from the index (0 or 'index') or
    columns (1 or 'columns').
index : single label or list-like
    Alternative to specifying axis (``labels, axis=0``
    is equivalent to ``index=labels``).
columns : single label or list-like
    Alternative to specifying axis (``labels, axis=1``
    is equivalent to ``columns=labels``).
level : int or level name, optional
    For MultiIndex, level from which the labels will be removed.
inplace : bool, default False
    If False, return a copy. Otherwise, do operation
    inplace and return None.
errors : {'ignore', 'raise'}, default 'raise'
    If 'ignore', suppress error and only existing labels are
    dropped.

Returns
-------
DataFrame
    DataFrame without the removed index or column labels.

Raises
------
KeyError
    If any of the labels is not found in the selected axis.

See Also
--------
DataFrame.loc : Label-location based indexer for selection by label.
DataFrame.dropna : Return DataFrame with labels on given axis omitted
    where (all or any) data are missing.
DataFrame.drop_duplicates : Return DataFrame with duplicate rows
    removed, optionally only considering certain columns.
Series.drop : Return Series with specified index labels removed.

Examples
--------
>>> df = pd.DataFrame(np.arange(12).reshape(3, 4),
...                   columns=['A', 'B', 'C', 'D'])
>>> df
   A  B   C   D
0  0  1   2   3
1  4  5   6   7
2  8  9  10  11

Drop columns

>>> df.drop(['B', 'C'], axis=1)
   A   D
0  0   3
1  4   7
2  8  11

>>> df.drop(columns=['B', 'C'])
   A   D
0  0   3
1  4   7
2  8  11

Drop a row by index

>>> df.drop([0, 1])
   A  B   C   D
2  8  9  10  11
```

```
Drop columns and/or rows of MultiIndex DataFrame

>>> midx = pd.MultiIndex(levels=[['lama', 'cow', 'falcon'],
...                              ['speed', 'weight', 'length']],
...                      codes=[[0, 0, 0, 1, 1, 1, 2, 2, 2],
...                             [0, 1, 2, 0, 1, 2, 0, 1, 2]])
>>> df = pd.DataFrame(index=midx, columns=['big', 'small'],
...                   data=[[45, 30], [200, 100], [1.5, 1], [30, 20],
...                         [250, 150], [1.5, 0.8], [320, 250],
...                         [1, 0.8], [0.3, 0.2]])
>>> df
                big     small
lama    speed   45.0    30.0
        weight  200.0   100.0
        length  1.5     1.0
cow     speed   30.0    20.0
        weight  250.0   150.0
        length  1.5     0.8
falcon  speed   320.0   250.0
        weight  1.0     0.8
        length  0.3     0.2

>>> df.drop(index='cow', columns='small')
                big
lama    speed   45.0
        weight  200.0
        length  1.5
falcon  speed   320.0
        weight  1.0
        length  0.3

>>> df.drop(index='length', level=1)
                big     small
lama    speed   45.0    30.0
        weight  200.0   100.0
cow     speed   30.0    20.0
        weight  250.0   150.0
falcon  speed   320.0   250.0
        weight  1.0     0.8
File:       /opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/pandas/core/frame.py
Type:       method
```

```
sals_df.drop(columns='comm')
```

	id	sal	active
0	1	1500.0	True
1	2	2000.0	True
2	3	2200.0	False

Note

We can also drop multiple columns by passing column names as list.

```
sals_df.drop(columns=['comm', 'active'])
```

	id	sal
0	1	1500.0
1	2	2000.0
2	3	2200.0

```
sals_df.drop(['comm', 'active'], axis=1)
```

	id	sal
0	1	1500.0
1	2	2000.0
2	3	2200.0

```
sals_df = sals_df.drop(columns='comm')
```

```
sals_df.columns
```

```
Index(['id', 'sal', 'active'], dtype='object')
```

```
sals_df.columns = ['employee_id', 'salary', 'commission']
```

```
sals_df
```

	employee_id	salary	commission
0	1	1500.0	True
1	2	2000.0	True
2	3	2200.0	False

```
sals_df.sort_index?
```

```
Signature:
sals_df.sort_index(
    axis=0,
    level=None,
    ascending:bool=True,
    inplace:bool=False,
    kind:str='quicksort',
    na_position:str='last',
    sort_remaining:bool=True,
    ignore_index:bool=False,
    key:Union[Callable[[_ForwardRef('Index')], Union[_ForwardRef('Index'), ~AnyArrayLike]],
NoneType]=None,
)
Docstring:
Sort object by labels (along an axis).

Returns a new DataFrame sorted by label if `inplace` argument is
``False``, otherwise updates the original DataFrame and returns None.

Parameters
----------
axis : {0 or 'index', 1 or 'columns'}, default 0
    The axis along which to sort.  The value 0 identifies the rows,
    and 1 identifies the columns.
level : int or level name or list of ints or list of level names
    If not None, sort on values in specified index level(s).
ascending : bool or list of bools, default True
    Sort ascending vs. descending. When the index is a MultiIndex the
    sort direction can be controlled for each level individually.
inplace : bool, default False
    If True, perform operation in-place.
kind : {'quicksort', 'mergesort', 'heapsort'}, default 'quicksort'
    Choice of sorting algorithm. See also ndarray.np.sort for more
    information.  `mergesort` is the only stable algorithm. For
    DataFrames, this option is only applied when sorting on a single
    column or label.
na_position : {'first', 'last'}, default 'last'
    Puts NaNs at the beginning if `first`; `last` puts NaNs at the end.
    Not implemented for MultiIndex.
sort_remaining : bool, default True
    If True and sorting by level and index is multilevel, sort by other
    levels too (in order) after sorting by specified level.
ignore_index : bool, default False
    If True, the resulting axis will be labeled 0, 1, …, n - 1.

    .. versionadded:: 1.0.0

key : callable, optional
    If not None, apply the key function to the index values
    before sorting. This is similar to the `key` argument in the
    builtin :meth:`sorted` function, with the notable difference that
    this `key` function should be *vectorized*. It should expect an
    ``Index`` and return an ``Index`` of the same shape. For MultiIndex
    inputs, the key is applied *per level*.

    .. versionadded:: 1.1.0

Returns
-------
DataFrame
    The original DataFrame sorted by the labels.

See Also
--------
Series.sort_index : Sort Series by the index.
DataFrame.sort_values : Sort DataFrame by the value.
Series.sort_values : Sort Series by the value.

Examples
--------
>>> df = pd.DataFrame([1, 2, 3, 4, 5], index=[100, 29, 234, 1, 150],
...                   columns=['A'])
>>> df.sort_index()
     A
1    4
29   2
100  1
150  5
234  3

By default, it sorts in ascending order, to sort in descending order,
use ``ascending=False``

>>> df.sort_index(ascending=False)
     A
234  3
150  5
100  1
```

```
29   2
1    4

A key function can be specified which is applied to the index before
sorting. For a ``MultiIndex`` this is applied to each level separately.

>>> df = pd.DataFrame({"a": [1, 2, 3, 4]}, index=['A', 'b', 'C', 'd'])
>>> df.sort_index(key=lambda x: x.str.lower())
   a
A  1
b  2
C  3
d  4
File:      /opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/pandas/core/frame.py
Type:      method
```

```
sals_df.sort_index(ascending=False)
```

	employee_id	salary	commission
0	1	1500.0	True
1	2	2000.0	True
2	3	2200.0	False

```
sals_df.sort_values?
```

```
Signature:
sals_df.sort_values(
    by,
    axis=0,
    ascending=True,
    inplace=False,
    kind='quicksort',
    na_position='last',
    ignore_index=False,
    key:Union[Callable[[_ForwardRef('Series')], Union[_ForwardRef('Series'), ~AnyArrayLike]],
NoneType]=None,
)
Docstring:
Sort by the values along either axis.

Parameters
----------
        by : str or list of str
            Name or list of names to sort by.

            - if `axis` is 0 or `'index'` then `by` may contain index
              levels and/or column labels.
            - if `axis` is 1 or `'columns'` then `by` may contain column
              levels and/or index labels.

            .. versionchanged:: 0.23.0

               Allow specifying index or column level names.
axis : {0 or 'index', 1 or 'columns'}, default 0
     Axis to be sorted.
ascending : bool or list of bool, default True
     Sort ascending vs. descending. Specify list for multiple sort
     orders.  If this is a list of bools, must match the length of
     the by.
inplace : bool, default False
     If True, perform operation in-place.
kind : {'quicksort', 'mergesort', 'heapsort'}, default 'quicksort'
     Choice of sorting algorithm. See also ndarray.np.sort for more
     information.  `mergesort` is the only stable algorithm. For
     DataFrames, this option is only applied when sorting on a single
     column or label.
na_position : {'first', 'last'}, default 'last'
     Puts NaNs at the beginning if `first`; `last` puts NaNs at the
     end.
ignore_index : bool, default False
     If True, the resulting axis will be labeled 0, 1, …, n - 1.

     .. versionadded:: 1.0.0

key : callable, optional
    Apply the key function to the values
    before sorting. This is similar to the `key` argument in the
    builtin :meth:`sorted` function, with the notable difference that
    this `key` function should be *vectorized*. It should expect a
    ``Series`` and return a Series with the same shape as the input.
    It will be applied to each column in `by` independently.

    .. versionadded:: 1.1.0

Returns
-------
DataFrame or None
    DataFrame with sorted values if inplace=False, None otherwise.

See Also
--------
DataFrame.sort_index : Sort a DataFrame by the index.
Series.sort_values : Similar method for a Series.

Examples
--------
>>> df = pd.DataFrame({
...     'col1': ['A', 'A', 'B', np.nan, 'D', 'C'],
...     'col2': [2, 1, 9, 8, 7, 4],
...     'col3': [0, 1, 9, 4, 2, 3],
...     'col4': ['a', 'B', 'c', 'D', 'e', 'F']
... })
>>> df
  col1  col2  col3 col4
0    A     2     0    a
1    A     1     1    B
2    B     9     9    c
3  NaN     8     4    D
4    D     7     2    e
5    C     4     3    F

Sort by col1
```

```
>>> df.sort_values(by=['col1'])
  col1  col2  col3 col4
0    A     2     0    a
1    A     1     1    B
2    B     9     9    c
5    C     4     3    F
4    D     7     2    e
3  NaN     8     4    D

Sort by multiple columns

>>> df.sort_values(by=['col1', 'col2'])
  col1  col2  col3 col4
1    A     1     1    B
0    A     2     0    a
2    B     9     9    c
5    C     4     3    F
4    D     7     2    e
3  NaN     8     4    D

Sort Descending

>>> df.sort_values(by='col1', ascending=False)
  col1  col2  col3 col4
4    D     7     2    e
5    C     4     3    F
2    B     9     9    c
0    A     2     0    a
1    A     1     1    B
3  NaN     8     4    D

Putting NAs first

>>> df.sort_values(by='col1', ascending=False, na_position='first')
  col1  col2  col3 col4
3  NaN     8     4    D
4    D     7     2    e
5    C     4     3    F
2    B     9     9    c
0    A     2     0    a
1    A     1     1    B

Sorting with a key function

>>> df.sort_values(by='col4', key=lambda col: col.str.lower())
   col1  col2  col3 col4
0    A     2     0    a
1    A     1     1    B
2    B     9     9    c
3  NaN     8     4    D
4    D     7     2    e
5    C     4     3    F
File:      /opt/anaconda3/envs/beakerx/lib/python3.6/site-packages/pandas/core/frame.py
Type:      method
```

```
sals_df.sort_values(by='employee_id', ascending=False)
```

	employee_id	salary	commission
2	3	2200.0	False
1	2	2000.0	True
0	1	1500.0	True

```
sals_df.sort_values(by='salary')
```

	employee_id	salary	commission
0	1	1500.0	True
1	2	2000.0	True
2	3	2200.0	False

```
sals_df.sort_values(by='salary', ascending=False)
```

	employee_id	salary	commission
2	3	2200.0	False
1	2	2000.0	True
0	1	1500.0	True

```
sals_ld = [
    {'id': 1, 'sal': 1500.0},
    {'id': 2, 'sal': 2000.0, 'comm': 10.0},
    {'id': 3, 'sal': 2200.0, 'active': False},
    {'id': 4, 'sal': 2000.0}
]
```

```
sals_df = pd.DataFrame(sals_ld)
```

```
sals_df.sort_values(by=['sal', 'id'])
```

	id	sal	comm	active
0	1	1500.0	NaN	NaN
1	2	2000.0	10.0	NaN
3	4	2000.0	NaN	NaN
2	3	2200.0	NaN	False

```
sals_df.sort_values(by=['sal', 'id'], ascending=[False, True])
```

	id	sal	comm	active
2	3	2200.0	NaN	False
1	2	2000.0	10.0	NaN
3	4	2000.0	NaN	NaN
0	1	1500.0	NaN	NaN