Learning_Pandas_Part_2_BasicAttributes

June 20, 2021

0.0.1 Prepared by Abhishek Kumar

0.0.2 https://www.linkedin.com/in/abhishekkumar-0311/

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# To get multiple outputs in the same cell

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

%matplotlib inline
```

[3]:
```
# Setup : DataFrame creation

salary = [['1','Abhishek Kumar','AIML', 'Machine Learning Engineer','M', 'Y',␣
 ↪'04051990', 1121000],
          ['2','Arjun Kumar','DM', 'Tech Lead','M', 'Y', '09031992', 109000],
          ['3','Vivek Raj','DM', 'Devops Engineer','M', 'N', np.NaN , 827000],
          ['4','Mika Singh','DM', 'Data Analyst','F', 'Y', '15101991',  np.NaN],
          ['5','Anusha Yenduri','AIML', 'Data Scientist','M', 'Y', '01011989', ␣
 ↪921000],
          ['6','Ritesh Srivastava','AIML', 'Data Engineer','M', 'Y', np.NaN,␣
 ↪785000]]

columns_name=['Emp_Id','Emp_Name','Department','Role','Gender', 'WFH Status',␣
 ↪'DOB', 'Salary']

emp_df = pd.DataFrame(salary,columns=columns_name)
emp_df
```

[3]:

	Emp_Id	Emp_Name	Department	Role	Gender	\
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	
1	2	Arjun Kumar	DM	Tech Lead	M	
2	3	Vivek Raj	DM	Devops Engineer	M	
3	4	Mika Singh	DM	Data Analyst	F	

```
4      5     Anusha Yenduri       AIML             Data Scientist      M
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N       NaN   827000.0
3          Y  15101991        NaN
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

0.1 A. Attributes and Underlying Data

1. df.index
2. df.columns
3. df.dtypes
4. df.ndim
5. df.shape
6. df.size
7. len(df)
8. df.axes
9. df.memory_usage()
10. df.info()
11. df.empty
12. df.values
13. df.select_dtypes()

[4]:
```
# Setup : DataFrame creation
emp_df_1a = emp_df.copy()
#emp_df_1a

# The index (row labels) of the DataFrame.
emp_df_1a.index

# The column labels of the DataFrame.
emp_df_1a.columns

# Returns a Series with the data type of each column.
# Columns with mixed types are stored with the object dtype.
emp_df_1a.dtypes
emp_df_1a.dtypes.value_counts()
```

[4]:
```
RangeIndex(start=0, stop=6, step=1)
```

[4]:
```
Index(['Emp_Id', 'Emp_Name', 'Department', 'Role', 'Gender', 'WFH Status',
       'DOB', 'Salary'],
      dtype='object')
```

[4]:
```
Emp_Id          object
Emp_Name        object
Department      object
Role            object
Gender          object
WFH Status      object
DOB             object
Salary         float64
dtype: object
```

[4]:
```
object     7
float64    1
dtype: int64
```

[5]:
```
# Return an int representing the number of axes / array dimensions.
emp_df_1a.ndim

# Return a tuple representing the dimensionality of the DataFrame.
emp_df_1a.shape

# Return an int representing the number of elements in this object.
# Return the number of rows if Series. Otherwise return the number of rows␣
 ↪times number of columns if DataFrame.
emp_df_1a.size

# No of rows
len(emp_df_1a)
```

[5]:
```
2
```

[5]:
```
(6, 8)
```

[5]:
```
48
```

[5]:
```
6
```

[6]:
```
# Return a list representing the axes of the DataFrame.
emp_df_1a.axes

emp_df_1a.axes[0]
emp_df_1a.axes[0][0]
emp_df_1a.axes[1]
len(emp_df_1a.axes[1])
emp_df_1a.axes[1][0]
```

[6]:
```
[RangeIndex(start=0, stop=6, step=1),
 Index(['Emp_Id', 'Emp_Name', 'Department', 'Role', 'Gender', 'WFH Status',
```

```
       'DOB', 'Salary'],
      dtype='object')]
```

[6]:
```
RangeIndex(start=0, stop=6, step=1)
```

[6]:
```
0
```

[6]:
```
Index(['Emp_Id', 'Emp_Name', 'Department', 'Role', 'Gender', 'WFH Status',
       'DOB', 'Salary'],
      dtype='object')
```

[6]:
```
8
```

[6]:
```
'Emp_Id'
```

DataFrame.memory_usage(self, index=True, deep=False) Return the memory usage of each column in bytes.

The memory usage can optionally include the contribution of the index and elements of object dtype. This value is displayed in DataFrame.info by default. This can be suppressed by setting pandas.options.display.memory_usage to False.

1. Parameters - index bool, default True : Specifies whether to include the memory usage of the DataFrame's index in returned Series. If index=True, the memory usage of the index is the first item in the output.
2. deep - bool, default False : If True, introspect the data deeply by interrogating object dtypes for system-level memory consumption, and include it in the returned values.
3. Returns - Series : A Series whose index is the original column names and whose values is the memory usage of each column in bytes.

[7]:
```
emp_df_1a.memory_usage(index=True)
emp_df_1a.memory_usage().max()
```

[7]:
```
Index          128
Emp_Id          48
Emp_Name        48
Department      48
Role            48
Gender          48
WFH Status      48
DOB             48
Salary          48
dtype: int64
```

[7]:
```
128
```

[8]:
```
emp_df_1a.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Emp_Id      6 non-null      object
 1   Emp_Name    6 non-null      object
 2   Department  6 non-null      object
 3   Role        6 non-null      object
 4   Gender      6 non-null      object
 5   WFH Status  6 non-null      object
 6   DOB         4 non-null      object
 7   Salary      5 non-null      float64
dtypes: float64(1), object(7)
memory usage: 512.0+ bytes
```

DataFrame.empty

0. Indicator whether DataFrame is empty.
1. True if DataFrame is entirely empty (no items), meaning any of the axes are of length 0.
2. Returns - bool : If DataFrame is empty, return True, if not return False.

Note : If DataFrame contains only NaNs, it is still not considered empty.

[9]:
```
emp_df_1a.empty
```

[9]:
```
False
```

DataFrame.values Return a Numpy representation of the DataFrame.

1. Only the values in the DataFrame will be returned, the axes labels will be removed.
2. Returns - numpy.ndarray : The values of the DataFrame.

Note : "DataFrame.to_numpy( )" is RECOMMENDED instead.

[10]:
```
emp_df_1a.values

emp_df_1a.Emp_Name.values
```

[10]:
```
array([['1', 'Abhishek Kumar', 'AIML', 'Machine Learning Engineer', 'M',
        'Y', '04051990', 1121000.0],
       ['2', 'Arjun Kumar', 'DM', 'Tech Lead', 'M', 'Y', '09031992',
        109000.0],
       ['3', 'Vivek Raj', 'DM', 'Devops Engineer', 'M', 'N', nan,
        827000.0],
       ['4', 'Mika Singh', 'DM', 'Data Analyst', 'F', 'Y', '15101991',
        nan],
       ['5', 'Anusha Yenduri', 'AIML', 'Data Scientist', 'M', 'Y',
```

```
        '01011989', 921000.0],
       ['6', 'Ritesh Srivastava', 'AIML', 'Data Engineer', 'M', 'Y', nan,
        785000.0]], dtype=object)
```

[10]:
```
array(['Abhishek Kumar', 'Arjun Kumar', 'Vivek Raj', 'Mika Singh',
       'Anusha Yenduri', 'Ritesh Srivastava'], dtype=object)
```

DataFrame.select_dtypes(self, include=None, exclude=None) Return a subset of the DataFrame's columns based on the column dtypes.

1. Parameters - include, exclude scalar or list-like : A selection of dtypes or strings to be included/excluded. At least one of these parameters must be supplied.
2. Returns - DataFrame : The subset of the frame including the dtypes in include and excluding the dtypes in exclude.
3. Raises - ValueError :

   If both of include and exclude are empty

   If include and exclude have overlapping elements

   If any kind of string dtype is passed in.

Notes:

```
To select all numeric types, use np.number or 'number'

To select strings you must use the object dtype, but note that this will return all object dtyp

See the numpy dtype hierarchy

To select datetimes, use np.datetime64, 'datetime' or 'datetime64'

To select timedeltas, use np.timedelta64, 'timedelta' or 'timedelta64'

To select Pandas categorical dtypes, use 'category'

To select Pandas datetimetz dtypes, use 'datetimetz' (new in 0.20.0) or 'datetime64[ns, tz]'
```

[11]:
```
emp_df_1a.select_dtypes(include = 'number')
```

[11]:

	Salary
0	1121000.0
1	109000.0
2	827000.0
3	NaN
4	921000.0
5	785000.0

[12]:
```
emp_df_1a.select_dtypes(include = ['O'])
```

[12]:
```
   Emp_Id           Emp_Name Department                        Role Gender  \
0       1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1       2        Arjun Kumar         DM                  Tech Lead      M
2       3          Vivek Raj         DM            Devops Engineer      M
3       4         Mika Singh         DM               Data Analyst      F
4       5     Anusha Yenduri       AIML             Data Scientist      M
5       6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB
0          Y  04051990
1          Y  09031992
2          N       NaN
3          Y  15101991
4          Y  01011989
5          Y       NaN
```

[13]:
```
emp_df_1a.select_dtypes(exclude = 'category')
```

[13]:
```
   Emp_Id           Emp_Name Department                        Role Gender  \
0       1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1       2        Arjun Kumar         DM                  Tech Lead      M
2       3          Vivek Raj         DM            Devops Engineer      M
3       4         Mika Singh         DM               Data Analyst      F
4       5     Anusha Yenduri       AIML             Data Scientist      M
5       6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N       NaN   827000.0
3          Y  15101991        NaN
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

0.2 B. Conversion

1. DataFrame.astype(self, dtype, copy, errors) - Cast a pandas object to a specified dtype dtype.
2. DataFrame.convert_dtypes(self, …) - Convert columns to best possible dtypes using dtypes supporting pd.NA.
3. DataFrame.copy(self, deep) - Make a copy of this object's indices and data.

0.2.1 1. DataFrame.astype(self: ~FrameOrSeries, dtype, copy: bool = True, errors: str = 'raise')

Cast a pandas object to a specified dtype dtype.

1. Parameters - dtypedata type, or dict of column name -> data type

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

2. copy - bool, default True : Return a copy when copy=True (be very careful setting copy=False as changes to values then may propagate to other pandas objects).
3. errors - {'raise', 'ignore'}, default 'raise' : Control raising of exceptions on invalid data for provided dtype.

   raise : allow exceptions to be raised

   ignore : suppress exceptions. On error return original object.

4. Returns - casted same type as caller

[14]:
```
# Setup : DataFrame creation
emp_df_2a = emp_df.copy()
# emp_df_2a

emp_df_2a.dtypes
emp_df_2a.astype('object').dtypes
emp_df_2a.astype({'Gender':'category', 'Emp_Id':'int8'}).dtypes
emp_df_2a.Gender.astype('category', copy=False).dtypes
```

[14]:
```
Emp_Id          object
Emp_Name        object
Department      object
Role            object
Gender          object
WFH Status      object
DOB             object
Salary         float64
dtype: object
```

[14]:
```
Emp_Id          object
Emp_Name        object
Department      object
Role            object
Gender          object
WFH Status      object
DOB             object
Salary          object
dtype: object
```

[14]:
```
Emp_Id             int8
Emp_Name         object
Department       object
Role             object
Gender         category
```

```
WFH Status      object
DOB             object
Salary         float64
dtype: object
```

[14]:
```
CategoricalDtype(categories=['F', 'M'], ordered=False)
```

0.2.2 2. DataFrame.convert_dtypes(self: ~FrameOrSeries, infer_objects: bool = True, convert_string: bool = True, convert_integer: bool = True, convert_boolean: bool = True)

Convert columns to best possible dtypes using dtypes supporting pd.NA. New in version 1.0.0. (check version with pd.___version___)

[15]:
```
# Setup : DataFrame creation
# emp_df_2b = emp_df.copy()
# emp_df_2b

# emp_df.dtypes
# emp_df_2b.convert_dtypes()
# emp_df_2b.dtypes
```

0.2.3 3. DataFrame.copy(self: ~FrameOrSeries, deep: bool = True)

Make a copy of this object's indices and data.

```
When deep=True (default), a new object will be created with a copy of the calling object's data

When deep=False, a new object will be created without copying the calling object's data or inde
```

1. Parameters - deep bool, default True : Make a deep copy, including a copy of the data and the indices. With deep=False neither the indices nor the data are copied.
2. Returns - copy Series or DataFrame : Object type matches caller.

Notes

```
When deep=True, data is copied but actual Python objects will not be copied recursively, only t

While Index objects are copied when deep=True, the underlying numpy array is not copied for per
```

[16]:
```
# Setup : DataFrame creation
emp_df_2c = emp_df.copy()
#emp_df_2c

# This created an new and independent copy of emp_df
# Changes to one dataframe does not impact or get reflected in another Dataframe
```

0.3 C. Descriptive Stats 1

There are many methods for Descriptive and Computation statistics. In this intro, i have taken up few useful ones.

1. df.nunique
2. df.count
3. df.all
4. df.any
5. df.describe

0.3.1 1. DataFrame.nunique(self, axis=0, dropna=True)

Count distinct observations over requested axis.

1. Parameters - axis{0 or 'index', 1 or 'columns'}, default 0 : The axis to use. 0 or 'index' for row-wise, 1 or 'columns' for column-wise.
2. dropna - bool, default True : Don't include NaN in the counts.
3. Returns - Series : Return Series with number of distinct observations. Can ignore NaN values.

[17]:
```
# Setup : DataFrame creation
emp_df_3a = emp_df.copy()
emp_df_3a

# To get unique values along axis = index/0
emp_df_3a.nunique()

# To get unique values for DOB along axis = index/0, including missing values
emp_df_3a.DOB.nunique(dropna=False)

# To get unique count for 1st column
emp_df_3a.nunique()[0]

# To get maximum 'unique count' among all columns
emp_df_3a.nunique().max()

# To get unique count as a numpy.ndarray
emp_df_3a.nunique().values

# To get unique count as a list
emp_df_3a.nunique().tolist()

# To get unique values along axis = columns/1
emp_df_3a.nunique(axis=1)
```

[17]:

	Emp_Id	Emp_Name	Department	Role	Gender	\
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	
1	2	Arjun Kumar	DM	Tech Lead	M	
2	3	Vivek Raj	DM	Devops Engineer	M	

```
3      4         Mika Singh         DM              Data Analyst      F
4      5     Anusha Yenduri       AIML            Data Scientist      M
5      6  Ritesh Srivastava       AIML             Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N       NaN   827000.0
3          Y  15101991        NaN
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

```
[17]: Emp_Id        6
      Emp_Name      6
      Department    2
      Role          6
      Gender        2
      WFH Status    2
      DOB           4
      Salary        5
      dtype: int64
```

```
[17]: 5
```

```
[17]: 6
```

```
[17]: 6
```

```
[17]: array([6, 6, 2, 6, 2, 2, 4, 5], dtype=int64)
```

```
[17]: [6, 6, 2, 6, 2, 2, 4, 5]
```

```
[17]: 0    8
      1    8
      2    7
      3    7
      4    8
      5    7
      dtype: int64
```

0.3.2 2. DataFrame.count(self, axis=0, level=None, numeric_only=False)

1. Count non-NA cells for each column or row.
2. The values None, NaN, NaT, and optionally numpy.inf (depending on pandas.options.mode.use_inf_as_na) are considered NA.
3. Returns - Series or DataFrame : For each column/row the number of non-NA/null entries. If level is specified returns a DataFrame.

[18]:
```
# Setup : DataFrame creation
emp_df_3b = emp_df.copy()
emp_df_3b

emp_df_3b.count(axis=0)

emp_df_3b.count(axis=1)
```

[18]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	NaN
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	NaN	785000.0

[18]:
```
Emp_Id        6
Emp_Name      6
Department    6
Role          6
Gender        6
WFH Status    6
DOB           4
Salary        5
dtype: int64
```

[18]:
```
0    8
1    8
2    7
3    7
4    8
5    7
dtype: int64
```

0.3.3 3. DataFrame.all(self, axis=0, bool_only=None, skipna=True, level=None, **kwargs)

Return whether all elements are True, potentially over an axis.

0. Returns True unless there at least one element within a series or along a Dataframe axis that

is False or equivalent (e.g. zero or empty).

1. Parameters - axis {0 or 'index', 1 or 'columns', None}, default 0 : Indicate which axis or axes should be reduced.

0 / 'index' : reduce the index, return a Series whose index is the original column labels. 1 / 'columns' : reduce the columns, return a Series whose index is the original index. None : reduce all axes, return a scalar.

2. bool_only - bool, default None : Include only boolean columns. If None, will attempt to use everything, then use only boolean data. Not implemented for Series.
3. skipna - bool, default True : Exclude NA/null values. If the entire row/column is NA and skipna is True, then the result will be True, as for an empty row/column. If skipna is False, then NA are treated as True, because these are not equal to zero.
4. level - int or level name, default None : If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series.
5. **kwargs - any, default None : Additional keywords have no effect but might be accepted for compatibility with NumPy.
6. Returns - Series or DataFrame : If level is specified, then, DataFrame is returned; otherwise, Series is returned.

[19]:
```
# Setup : DataFrame creation
emp_df_3c = emp_df.copy()
emp_df_3c

#
emp_df_3c.all()

# Comparison of dataframe to return a Boolean value
(emp_df_3c == emp_df).all()
(emp_df_3c == emp_df).all(axis= None)

# NaNs are also considered to be equal and hence return TRUE
emp_df_3c.equals(emp_df)
```

[19]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0
3	4	Mika Singh	DM	Data Analyst	F			
4	5	Anusha Yenduri	AIML	Data Scientist	M			
5	6	Ritesh Srivastava	AIML	Data Engineer	M			

```
3          Y  15101991        NaN
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

```
[19]: Emp_Id         True
      Emp_Name       True
      Department     True
      Role           True
      Gender         True
      WFH Status     True
      DOB            True
      Salary         True
      dtype: bool
```

```
[19]: Emp_Id          True
      Emp_Name        True
      Department      True
      Role            True
      Gender          True
      WFH Status      True
      DOB            False
      Salary         False
      dtype: bool
```

```
[19]: False
```

```
[19]: True
```

0.3.4 4. DataFrame.any(self, axis=0, bool_only=None, skipna=True, level=None, **kwargs)

Return whether any element is True, potentially over an axis.

0. Returns False unless there at least one element within a series or along a Dataframe axis that is True or equivalent (e.g. non-zero or non-empty).
1. Parameters - axis {0 or 'index', 1 or 'columns', None}, default 0 : Indicate which axis or axes should be reduced.

0 / 'index' : reduce the index, return a Series whose index is the original column labels. 1 / 'columns' : reduce the columns, return a Series whose index is the original index. None : reduce all axes, return a scalar.

2. bool_only - bool, default None : Include only boolean columns. If None, will attempt to use everything, then use only boolean data. Not implemented for Series.
3. skipna - bool, default True : Exclude NA/null values. If the entire row/column is NA and skipna is True, then the result will be False, as for an empty row/column. If skipna is False, then NA are treated as True, because these are not equal to zero.

4. level - int or level name, default None : If the axis is a MultiIndex (hierarchical), count along a particular level, collapsing into a Series.
5. **kwargs - any, default None : Additional keywords have no effect but might be accepted for compatibility with NumPy.
6. Returns - Series or DataFrame : If level is specified, then, DataFrame is returned; otherwise, Series is returned.

[20]:
```
# Setup : DataFrame creation
emp_df_3d = emp_df.copy()
emp_df_3d

# Check if any 'True' exists in given axis
emp_df_3d.any()
(~emp_df_3d.any()).tolist()

emp_df_3d.any(axis=None)
```

[20]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	NaN
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	NaN	785000.0

[20]:
```
Emp_Id        True
Emp_Name      True
Department    True
Role          True
Gender        True
WFH Status    True
DOB           True
Salary        True
dtype: bool
```

[20]:
```
[False, False, False, False, False, False, False, False]
```

[20]:
```
True
```

0.3.5 5. DataFrame.describe(self: ~FrameOrSeries, percentiles=None, include=None, exclude=None)

Generate descriptive statistics.

```
Descriptive statistics include those that summarize the central tendency, dispersion and shape

Analyzes both numeric and object series, as well as DataFrame column sets of mixed data types.
```

1. Parameters - percentiles list-like of numbers, optional : The percentiles to include in the output. All should fall between 0 and 1. The default is [.25, .5, .75], which returns the 25th, 50th, and 75th percentiles.
2. include - 'all', list-like of dtypes or None (default), optional : A white list of data types to include in the result. Ignored for Series.

Here are the options:

```
'all' : All columns of the input will be included in the output.

A list-like of dtypes : Limits the results to the provided data types. To limit the result to r

None (default) : The result will include all numeric columns.
```

3. exclude - list-like of dtypes or None (default), optional, : A black list of data types to omit from the result. Ignored for Series. Options are same as include.
4. Returns - Series or DataFrame : Summary statistics of the Series or Dataframe provided.

[21]:
```
# Setup : DataFrame creation
emp_df_3e = emp_df.copy()
#emp_df_3e

# By default, generate stats for only numeric columns
emp_df_3e.describe()

# Generating stats for all columns
emp_df_3e.describe(include= 'all')
```

[21]:
```
              Salary
count  5.000000e+00
mean   7.526000e+05
std    3.823883e+05
min    1.090000e+05
25%    7.850000e+05
50%    8.270000e+05
75%    9.210000e+05
max    1.121000e+06
```

[21]:
```
        Emp_Id    Emp_Name Department                        Role Gender  \
count        6           6          6                           6      6
```

```
unique      6           6          2                           6      2
top         2  Mika Singh         DM  Machine Learning Engineer      M
freq        1           1          3                           1      5
mean      NaN         NaN        NaN                         NaN    NaN
std       NaN         NaN        NaN                         NaN    NaN
min       NaN         NaN        NaN                         NaN    NaN
25%       NaN         NaN        NaN                         NaN    NaN
50%       NaN         NaN        NaN                         NaN    NaN
75%       NaN         NaN        NaN                         NaN    NaN
max       NaN         NaN        NaN                         NaN    NaN
```

```
       WFH Status       DOB        Salary
count           6         4  5.000000e+00
unique          2         4           NaN
top             Y  15101991           NaN
freq            5         1           NaN
mean          NaN       NaN  7.526000e+05
std           NaN       NaN  3.823883e+05
min           NaN       NaN  1.090000e+05
25%           NaN       NaN  7.850000e+05
50%           NaN       NaN  8.270000e+05
75%           NaN       NaN  9.210000e+05
max           NaN       NaN  1.121000e+06
```

0.4 D. Handling Missing Data

NaN is the default missing value marker

1. df.isna() , df.isnull()
2. df.notna() , df.notnull()
3. df.fillna()
4. df.dropna()
5. df.replace()
6. df.interpolate() - skipped as of now

0.4.1 1. Detecting missing values

i. DataFrame.isna(self)

```
Detect missing values.

Return a boolean same-sized object indicating if the values are NA. NA values, such as None or
```

[22]:
```
# Setup : DataFrame creation
emp_df_4a = emp_df.copy()
emp_df_4a

emp_df_4a.isna()
```

[22]:
	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	NaN
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	NaN	785000.0

[22]:
	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	True	False
3	False	False	False	False	False	False	False	True
4	False	False	False	False	False	False	False	False
5	False	False	False	False	False	False	True	False

ii. DataFrame.notna(self)

Detect existing (non-missing) values.

Return a boolean same-sized object indicating if the values are not NA. Non-missing values get

[23]:
```
emp_df_4a.notna()
```

[23]:
	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	True	True	True	True	True	True	True	True
1	True	True	True	True	True	True	True	True
2	True	True	True	True	True	True	False	True
3	True	True	True	True	True	True	True	False
4	True	True	True	True	True	True	True	True
5	True	True	True	True	True	True	False	True

Pandas/NumPy uses the fact that np.nan != np.nan, and treats None like np.nan

[24]:
```
None == None
np.NaN == np.NaN
```

[24]: True

[24]: False

0.4.2 Important Notes:

For datetime64[ns] types, NaT represents missing values.

Pandas objects provide compatibility between NaT and NaN.

When summing data, NA (missing) values are treated as zero.

If the data are all NA, the result is 0.

Cumulative methods like cumsum() and cumprod() ignore NA values by default, but preserve them in the resulting arrays. To override this behaviour and include NA values, use skipna=False.

The sum of an empty or all-NA Series or column of a DataFrame is 0.

The product of an empty or all-NA Series or column of a DataFrame is 1.

NA groups in GroupBy are automatically excluded.

0.4.3 2. Filling Missing Values

i. Replace NA with a scalar value

ii. Fill gaps forward or backward

iii. Fill with a PandasObject

0.4.4 DataFrame.fillna(self, value=None, method=None, axis=None, inplace=False, limit=None, downcast=None)

Fill NA/NaN values using the specified method.

1. 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.
2. 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.
3. axis - {0 or 'index', 1 or 'columns'} : Axis along which to fill missing values.
4. 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).
5. 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.

6. 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).

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

i. Replace NA with a scalar value

[25]:
```
# Setup : DataFrame creation
emp_df_4b = emp_df.copy()
emp_df_4b

emp_df_4b.fillna(0)
emp_df_4b.fillna('missing')
```

[25]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	NaN
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	NaN	785000.0

[25]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	0	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	0.0
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	0	785000.0

[25]:
```
  Emp_Id           Emp_Name Department                       Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                  Tech Lead      M
2      3          Vivek Raj         DM            Devops Engineer      M
3      4         Mika Singh         DM               Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      M
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1.121e+06
1          Y  09031992     109000
2          N   missing     827000
3          Y  15101991    missing
4          Y  01011989     921000
5          Y   missing     785000
```

ii. Fill gaps forward or backward

[26]:
```
emp_df_4b
emp_df_4b.fillna(method = 'bfill')
emp_df_4b.fillna(method = 'ffill')
```

[26]:
```
  Emp_Id           Emp_Name Department                       Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                  Tech Lead      M
2      3          Vivek Raj         DM            Devops Engineer      M
3      4         Mika Singh         DM               Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      M
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N       NaN   827000.0
3          Y  15101991        NaN
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

[26]:
```
  Emp_Id           Emp_Name Department                       Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                  Tech Lead      M
2      3          Vivek Raj         DM            Devops Engineer      M
3      4         Mika Singh         DM               Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      M
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
```

```
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N  15101991   827000.0
3          Y  15101991   921000.0
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

[26]:
```
  Emp_Id           Emp_Name Department                        Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                   Tech Lead      M
2      3          Vivek Raj         DM            Devops Engineer      M
3      4         Mika Singh         DM               Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      M
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N  09031992   827000.0
3          Y  15101991   827000.0
4          Y  01011989   921000.0
5          Y  01011989   785000.0
```

iii. Fill with a PandasObject

[27]:
```
# Setup : DataFrame creation
emp_df_4b_2 = emp_df.copy()
emp_df_4b_2

emp_df_4b_2['FillingNaN'] = emp_df_4b_2.Salary.fillna(emp_df_4b_2.Salary.mean())
emp_df_4b_2
```

[27]:
```
  Emp_Id           Emp_Name Department                        Role Gender  \
0      1     Abhishek Kumar       AIML  Machine Learning Engineer      M
1      2        Arjun Kumar         DM                   Tech Lead      M
2      3          Vivek Raj         DM            Devops Engineer      M
3      4         Mika Singh         DM               Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      M
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N       NaN   827000.0
3          Y  15101991        NaN
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

[27]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary	FillingNaN
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	NaN	752600.0
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	NaN	785000.0	785000.0

[28]:
```
emp_df_4b_2['Salary'] = emp_df_4b_2.Salary.fillna(emp_df_4b_2.Salary.mean())
emp_df_4b_2

# Setup : DataFrame creation
emp_df_4b_2_ = emp_df.copy()
# emp_df_4b_2_

# Using inplace = True , enables to save the existing dF and also helps in␣
 ↪chaining
emp_df_4b_2_.Salary.fillna(emp_df_4b_2_.Salary.mean() , inplace=True)
emp_df_4b_2_
```

[28]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary	FillingNaN
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	752600.0	752600.0
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	NaN	785000.0	785000.0

[28]:

	Emp_Id	Emp_Name	Department	Role	Gender
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M
1	2	Arjun Kumar	DM	Tech Lead	M

```
2      3          Vivek Raj         DM            Devops Engineer      M
3      4         Mika Singh         DM               Data Analyst      F
4      5     Anusha Yenduri       AIML             Data Scientist      M
5      6  Ritesh Srivastava       AIML              Data Engineer      M

  WFH Status       DOB     Salary
0          Y  04051990  1121000.0
1          Y  09031992   109000.0
2          N       NaN   827000.0
3          Y  15101991   752600.0
4          Y  01011989   921000.0
5          Y       NaN   785000.0
```

0.4.5 3. Drop Missing Values

DataFrame.dropna(self, axis=0, how='any', thresh=None, subset=None, inplace=False)[source] Remove missing values.

1. Parameters - axis {0 or 'index', 1 or 'columns'}, default 0 : Determine if rows or columns which contain missing values are removed.

   0, or 'index' : Drop rows which contain missing values.

   1, or 'columns' : Drop columns which contain missing value.

   Changed in version 1.0.0: Pass tuple or list to drop on multiple axes. Only a single axis is allowed.

2. how - {'any', 'all'}, default 'any' : Determine if row or column is removed from DataFrame, when we have at least one NA or all NA.

   'any' : If any NA values are present, drop that row or column.

   'all' : If all values are NA, drop that row or column.

3. thresh - int, optional : Require that many non-NA values.

4. subset - array-like, optional : Labels along other axis to consider, e.g. if you are dropping rows these would be a list of columns to include.

5. inplace - bool, default False : If True, do operation inplace and return None.

6. Returns - DataFrame : DataFrame with NA entries dropped from it.

[29]:
```
# Setup : DataFrame creation
emp_df_4b_3 = emp_df.copy()
emp_df_4b_3

# Drop all the rows with any Nan value
emp_df_4b_3.dropna()

# Note the behaviour of subset= parameter. This defines the list of columns␣
 ↪that is looked for NaN
```

```
# axis = 0 checks for each record and drops rows which have NULLs for the 2 mentioned columns
emp_df_4b_3.dropna(subset = ['Emp_Id','DOB'], inplace = True, how = 'any', axis = 0)
emp_df_4b_3
```

[29]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
2	3	Vivek Raj	DM	Devops Engineer	M	N	NaN	827000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	NaN
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0
5	6	Ritesh Srivastava	AIML	Data Engineer	M	Y	NaN	785000.0

[29]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0

[29]:

	Emp_Id	Emp_Name	Department	Role	Gender	WFH Status	DOB	Salary
0	1	Abhishek Kumar	AIML	Machine Learning Engineer	M	Y	04051990	1121000.0
1	2	Arjun Kumar	DM	Tech Lead	M	Y	09031992	109000.0
3	4	Mika Singh	DM	Data Analyst	F	Y	15101991	NaN
4	5	Anusha Yenduri	AIML	Data Scientist	M	Y	01011989	921000.0

0.4.6 4. Fill missing values with REPLACE()

DataFrame.replace(self, to_replace=None, value=None, inplace=False, limit=None, regex=False, method='pad')[source] Replace values given in to_replace with value.

Values of the DataFrame are replaced with other values dynamically. This differs from updating with .loc or .iloc, which require you to specify a location to update with some value.

1. Parameters - to_replace str - regex, list, dict, Series, int, float, or None

   How to find the values that will be replaced.

   numeric, str or regex:

   ```
   numeric: numeric values equal to to_replace will be replaced with value

   str: string exactly matching to_replace will be replaced with value

   regex: regexs matching to_replace will be replaced with value
   ```

   list of str, regex, or numeric:

   ```
   First, if to_replace and value are both lists, they must be the same length.

   Second, if regex=True then all of the strings in both lists will be interpreted as regexs

   str, regex and numeric rules apply as above.
   ```

   dict:

   ```
   Dicts can be used to specify different replacement values for different existing values.

   For a DataFrame a dict can specify that different values should be replaced in different

   For a DataFrame nested dictionaries, e.g., {'a': {'b': np.nan}}, are read as follows: loc
   ```

   None:

   ```
   This means that the regex argument must be a string, compiled regular expression, or list
   ```

2. value - scalar, dict, list, str, regex, default None : Value to replace any values matching to_replace with. For a DataFrame a dict of values can be used to specify which value to use for each column (columns not in the dict will not be filled). Regular expressions, strings and lists or dicts of such objects are also allowed.

3. inplace - bool, default False : If True, in place. Note: this will modify any other views on this object (e.g. a column from a DataFrame). Returns the caller if this is True.

4. limit - int, default None : Maximum size gap to forward or backward fill.

5. regex - bool or same types as to_replace, default False : Whether to interpret to_replace and/or value as regular expressions. If this is True then to_replace must be a string. Alternatively, this could be a regular expression or a list, dict, or array of regular expressions in which case to_replace must be None.

6. method - {‘pad’, ‘ffill’, ‘bfill’, None} - The method to use when for replacement, when to_replace is a scalar, list or tuple and value is None.
7. Returns - DataFrame - Object after replacement.
8. Raises -

```
AssertionError - If regex is not a bool and to_replace is not None.
Type Error - If to_replace is a dict and value is not a list, dict, ndarray, or Series
ValueError - If a list or an ndarray is passed to to_replace and value but they are not t
```

0.4.7 For more details, refer - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.replace.html#pandas.DataFrame.replace

.replace( ) is a general replacement function and its usage can be extended in multiple ways.

0.5 Replicating COALESCE() of SQL,SAS

- COALESCE returns the first non-null value from a list of values.
- **FILLNA() can be used to implement this functionality.**
- **https://stackoverflow.com/questions/43177685/how-to-implement-sql-coalesce-in-pandas/43180501**
- **https://stackoverflow.com/questions/38152389/coalesce-values-from-2-columns-into-a-single-column-in-a-pandas-dataframe/38152458**
- **https://kanoki.org/2019/08/17/pandas-coalesce-replace-value-from-another-column/**

[72]:
```
sample = {
'col_a':['Houston,TX', np.NaN, 'Chicago,IL', 'Phoenix,AZ',      'San Diego,CA'],
'col_b':['62K-70K', '71K-78K', '69K-76K', '62K-72K', '71K-78K' ],
'col_c':['A','XYZ','A','a','c'],
'col_d':['  1x', np.NaN, np.NaN, '1x', np.NaN]
}
df_sample = pd.DataFrame(sample)
df_sample
df = df_sample.copy()
```

[72]:

	col_a	col_b	col_c	col_d
0	Houston,TX	62K-70K	A	1x
1	NaN	71K-78K	XYZ	NaN
2	Chicago,IL	69K-76K	A	NaN
3	Phoenix,AZ	62K-72K	a	1x
4	San Diego,CA	71K-78K	c	NaN

[73]:
```
# creating a new column 'coalesce' which takes 1st Non-Null value starting from
 ↪col_a , col_b, col_c, col_d (left to right)

df['coalesce'] = df.fillna(method='bfill', axis='columns').iloc[:, 0]
```

```
df
```

[73]:

	col_a	col_b	col_c	col_d	coalesce
0	Houston,TX	62K-70K	A	1x	Houston,TX
1	NaN	71K-78K	XYZ	NaN	71K-78K
2	Chicago,IL	69K-76K	A	NaN	Chicago,IL
3	Phoenix,AZ	62K-72K	a	1x	Phoenix,AZ
4	San Diego,CA	71K-78K	c	NaN	San Diego,CA

[74]:
```
df['LastNonNull'] = np.NaN
df['LastNonNull'] = df.fillna(method='ffill', axis='columns').iloc[:, 3]
df
```

[74]:

	col_a	col_b	col_c	col_d	coalesce	LastNonNull
0	Houston,TX	62K-70K	A	1x	Houston,TX	1x
1	NaN	71K-78K	XYZ	NaN	71K-78K	XYZ
2	Chicago,IL	69K-76K	A	NaN	Chicago,IL	A
3	Phoenix,AZ	62K-72K	a	1x	Phoenix,AZ	1x
4	San Diego,CA	71K-78K	c	NaN	San Diego,CA	c

[65]:
```
df['FirstNonNull'] = np.NaN
df['FirstNonNull'] = df.FirstNonNull.fillna(df.col_a).fillna(df.col_b).
↪fillna(df.col_c).fillna(df.col_d)
df
```

[65]:

	col_a	col_b	col_c	col_d	coalesce	LastNonNull	FirstNonNull
0	Houston,TX	62K-70K	A	1x	Houston,TX	1x	Houston,TX
1	69K-76K	NaN	NaN	NaN	69K-76K	69K-76K	69K-76K
2	Chicago,IL	69K-76K	A	NaN	Chicago,IL	A	Chicago,IL
3	Phoenix,AZ	62K-72K	a	1x	Phoenix,AZ	1x	Phoenix,AZ
4	San Diego,CA	71K-78K	c	NaN	San Diego,CA	c	San Diego,CA

[ ]:
```

```

[75]:
```
df['coalesce2'] = df.col_a.fillna(method='bfill')
df
```

[75]:

	col_a	col_b	col_c	col_d	coalesce	LastNonNull	coalesce2
0	Houston,TX	62K-70K	A	1x	Houston,TX	1x	Houston,TX
1	NaN	71K-78K	XYZ	NaN	71K-78K	XYZ	Chicago,IL
2	Chicago,IL	69K-76K	A	NaN	Chicago,IL	A	Chicago,IL
3	Phoenix,AZ	62K-72K	a	1x	Phoenix,AZ	1x	Phoenix,AZ
4	San Diego,CA	71K-78K	c	NaN	San Diego,CA	c	San Diego,CA

[ ]:
```

```