Pandas-DataFrame

December 22, 2023

1 Pandas DataFrame

1.1 Import Required Libraries

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

1.2 Create DataFrame

[3]:
```
# help(pd)
```

[5]:
```
# dir(pd)
```

[7]:
```
# help(pd.DataFrame)
```

[17]:
```
df = pd.DataFrame(
    data=[[1, 2, 3, 4, 5], [-2, 0, 4, -6, 3]],
    columns=pd.Index(
        data=list("abcde"),
        name="variables",
    ),
    index=pd.Index(data=["x", "y"], name="index"),
)
print(df)
```

```
variables  a  b  c  d  e
index
x          1  2  3  4  5
y         -2  0  4 -6  3
```

[13]:
```
df.columns
```

[13]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

[18]:
```
# help(pd.Index)
```

[19]:
```
df = pd.DataFrame(
    data=[
        ["Michael", "Male", 35, "Lecturer"],
        ["Lucy", "Female", 25, "Accountant"],
        ["Smith", "Male", 32, "Driver"],
        ["Andrea", "Female", 22, "Engineer"],
        ["Jane", "Female", 20, "Designer"],
    ]
)
print(df)
```

```
         0       1   2           3
0  Michael    Male  35    Lecturer
1     Lucy  Female  25  Accountant
2    Smith    Male  32      Driver
3   Andrea  Female  22    Engineer
4     Jane  Female  20    Designer
```

[20]:
```
df = pd.DataFrame(
    data=np.array(
        object=[
            ["Michael", "Male", 35, "Lecturer"],
            ["Lucy", "Female", 25, "Accountant"],
            ["Smith", "Male", 32, "Driver"],
            ["Andrea", "Female", 22, "Engineer"],
            ["Jane", "Female", 20, "Designer"],
        ]
    ),
    columns=pd.Index(
        name="Field",
        data=["Name", "Gender", "Age", "Job"],
    ),
    index=pd.Index(
        name="ID",
        data=["DA03", "DA06", "DA17", "DA12", "DA09"],
    ),
)
print(df)
```

```
Field     Name  Gender Age         Job
ID
DA03   Michael    Male  35    Lecturer
DA06      Lucy  Female  25  Accountant
DA17     Smith    Male  32      Driver
DA12    Andrea  Female  22    Engineer
DA09      Jane  Female  20    Designer
```

1.3 Access Elements

[22]:
```
df.loc["DA03", "Name"]
```

[22]:
```
'Michael'
```

[23]:
```
df.iloc[0, 0]
```

[23]:
```
'Michael'
```

1.3.1 Access First Row?

[24]:
```
df.loc["DA03", :]
```

[24]:
```
Field
Name       Michael
Gender        Male
Age             35
Job       Lecturer
Name: DA03, dtype: object
```

[26]:
```
type(df.loc["DA03", :])
```

[26]:
```
pandas.core.series.Series
```

[25]:
```
df.loc[["DA03"], :]
```

[25]:
```
Field     Name Gender Age       Job
ID
DA03   Michael   Male  35  Lecturer
```

[27]:
```
type(df.loc[["DA03"], :])
```

[27]:
```
pandas.core.frame.DataFrame
```

[28]:
```
df
```

[28]:
```
Field     Name  Gender Age         Job
ID
DA03   Michael    Male  35    Lecturer
DA06      Lucy  Female  25  Accountant
DA17     Smith    Male  32      Driver
DA12    Andrea  Female  22    Engineer
DA09      Jane  Female  20    Designer
```

1.3.2 Slice First and Third Row?

[29]:
```
df.loc[["DA03", "DA17"], :]
```

[29]:
```
Field     Name Gender Age       Job
ID
DA03   Michael   Male  35  Lecturer
DA17     Smith   Male  32    Driver
```

1.3.3 Access Column "Age"?

[31]:
```
df.loc[:, ["Age"]]
```

[31]:
```
Field Age
ID
DA03   35
DA06   25
DA17   32
DA12   22
DA09   20
```

[34]:
```
df.loc[["DA03"]]
```

[34]:
```
Field     Name Gender Age       Job
ID
DA03   Michael   Male  35  Lecturer
```

[36]:
```
df[["Age"]]
```

[36]:
```
Field Age
ID
DA03   35
DA06   25
DA17   32
DA12   22
DA09   20
```

1.4 Modify Elements

[37]:
```
df = pd.DataFrame(
    data=np.array(
        object=[
            ["Michael", "Male", 35, "Lecturer"],
            ["Lucy", "Female", 25, "Accountant"],
            ["Smith", "Male", 32, "Driver"],
            ["Andrea", "Female", 22, "Engineer"],
            ["Jane", "Female", 20, "Designer"],
        ]
```

```
    ),
    columns=pd.Index(
        name="Field",
        data=["Name", "Gender", "Age", "Job"],
    ),
    index=pd.Index(
        name="ID",
        data=["DA03", "DA06", "DA17", "DA12", "DA09"],
    ),
)
print(df)
```

```
Field     Name  Gender Age         Job
ID
DA03   Michael    Male  35    Lecturer
DA06      Lucy  Female  25  Accountant
DA17     Smith    Male  32      Driver
DA12    Andrea  Female  22    Engineer
DA09      Jane  Female  20    Designer
```

[42]:
```
df.columns.name = "Variables"
```

[43]:
```
df
```

[43]:
```
Variables     Name  Gender Age         Job
ID
DA03       Michael    Male  35    Lecturer
DA06          Lucy  Female  25  Accountant
DA17         Smith    Male  32      Driver
DA12        Andrea  Female  22    Engineer
DA09          Jane  Female  20    Designer
```

[46]:
```
df.index.name = "Index"
```

[47]:
```
df
```

[47]:
```
Variables     Name  Gender Age         Job
Index
DA03       Michael    Male  35    Lecturer
DA06          Lucy  Female  25  Accountant
DA17         Smith    Male  32      Driver
DA12        Andrea  Female  22    Engineer
DA09          Jane  Female  20    Designer
```

[50]:
```
df.rename(mapper={"Gender": "Sex"}, axis=1, inplace=True)
```

[51]:
```
df
```

[51]:
```
Variables     Name     Sex Age         Job
Index
DA03       Michael    Male  35    Lecturer
DA06          Lucy  Female  25  Accountant
DA17         Smith    Male  32      Driver
DA12        Andrea  Female  22    Engineer
DA09          Jane  Female  20    Designer
```

1.4.1 Rename "DA12" to "DA02"?

[53]:
```
df.rename(mapper={"DA12": "DA02"}, axis="index", inplace=True)
```

[54]:
```
df
```

[54]:
```
Variables     Name     Sex Age         Job
Index
DA03       Michael    Male  35    Lecturer
DA06          Lucy  Female  25  Accountant
DA17         Smith    Male  32      Driver
DA02        Andrea  Female  22    Engineer
DA09          Jane  Female  20    Designer
```

1.4.2 Change Age of Michael to "36"?

[57]:
```
df.loc["DA03", "Age"] = 36
df
```

[57]:
```
Variables     Name     Sex Age         Job
Index
DA03       Michael    Male  36    Lecturer
DA06          Lucy  Female  25  Accountant
DA17         Smith    Male  32      Driver
DA02        Andrea  Female  22    Engineer
DA09          Jane  Female  20    Designer
```

1.5 Add/Remove Row/Column

[58]:
```
df.loc["DA10"] = ["Say", "Male", 37, "Lecturer"]
df
```

[58]:
```
Variables     Name     Sex Age         Job
Index
DA03       Michael    Male  36    Lecturer
DA06          Lucy  Female  25  Accountant
DA17         Smith    Male  32      Driver
DA02        Andrea  Female  22    Engineer
DA09          Jane  Female  20    Designer
```

```
DA10          Say    Male  37    Lecturer
```

[62]:
```
df.drop(labels="DA10", axis=0, inplace=True)
```

[63]:
```
df
```

[63]:
```
Variables     Name     Sex Age         Job
Index
DA03       Michael    Male  36    Lecturer
DA06          Lucy  Female  25  Accountant
DA17         Smith    Male  32      Driver
DA02        Andrea  Female  22    Engineer
DA09          Jane  Female  20    Designer
```

[64]:
```
df["Score"] = [6, 5, 8, 0, 9]
```

[65]:
```
df
```

[65]:
```
Variables     Name     Sex Age         Job  Score
Index
DA03       Michael    Male  36    Lecturer      6
DA06          Lucy  Female  25  Accountant      5
DA17         Smith    Male  32      Driver      8
DA02        Andrea  Female  22    Engineer      0
DA09          Jane  Female  20    Designer      9
```

1.5.1 Drop Column "Score"?

[66]:
```
df = df.drop(labels=["Score"], axis="columns")
```

[67]:
```
df
```

[67]:
```
Variables     Name     Sex Age         Job
Index
DA03       Michael    Male  36    Lecturer
DA06          Lucy  Female  25  Accountant
DA17         Smith    Male  32      Driver
DA02        Andrea  Female  22    Engineer
DA09          Jane  Female  20    Designer
```

[69]:
```
df["Score"] = [6, 5, 8, 0, 9]
df
```

[69]:
```
Variables     Name     Sex Age         Job  Score
Index
DA03       Michael    Male  36    Lecturer      6
DA06          Lucy  Female  25  Accountant      5
DA17         Smith    Male  32      Driver      8
```

```
DA02        Andrea  Female  22    Engineer      0
DA09          Jane  Female  20    Designer      9
```

[74]:
```
# df.drop(labels=4, axis=1, inplace=True) # Error
```

1.6 Transformation

[119]:
```
df = pd.DataFrame(
    data=np.array(
        object=[
            ["Michael", "Male", 35, "Lecturer"],
            ["Lucy", "Female", 25, "Accountant"],
            ["Smith", "Male", 32, "Driver"],
            ["Andrea", "Female", 22, "Engineer"],
            ["Jane", "Female", 20, "Designer"],
        ]
    ),
    columns=pd.Index(
        name="Field",
        data=["Name", "Gender", "Age", "Job"],
    ),
    index=pd.Index(
        name="ID",
        data=["DA03", "DA06", "DA17", "DA12", "DA09"],
    ),
)
print(df)
```

```
Field     Name  Gender Age         Job
ID
DA03   Michael    Male  35    Lecturer
DA06      Lucy  Female  25  Accountant
DA17     Smith    Male  32      Driver
DA12    Andrea  Female  22    Engineer
DA09      Jane  Female  20    Designer
```

[79]:
```
df.sort_values(by=["Name"])
```

[79]:
```
Field     Name  Gender Age         Job
ID
DA12    Andrea  Female  22    Engineer
DA09      Jane  Female  20    Designer
DA06      Lucy  Female  25  Accountant
DA03   Michael    Male  35    Lecturer
DA17     Smith    Male  32      Driver
```

[83]:
```
df.sort_values(by=["Gender", "Age"], ascending=[False, True])
```

[83]:
```
Field     Name  Gender Age         Job
ID
DA17     Smith    Male  32      Driver
DA03   Michael    Male  35    Lecturer
DA09      Jane  Female  20    Designer
DA12    Andrea  Female  22    Engineer
DA06      Lucy  Female  25  Accountant
```

[90]:
```
df.sort_index(axis=0,ascending=False)
```

[90]:
```
Field     Name  Gender Age         Job
ID
DA17     Smith    Male  32      Driver
DA12    Andrea  Female  22    Engineer
DA09      Jane  Female  20    Designer
DA06      Lucy  Female  25  Accountant
DA03   Michael    Male  35    Lecturer
```

[92]:
```
df.sort_index().sort_index(axis="columns")
```

[92]:
```
Field Age  Gender         Job     Name
ID
DA03   35    Male    Lecturer  Michael
DA06   25  Female  Accountant     Lucy
DA09   20  Female    Designer     Jane
DA12   22  Female    Engineer   Andrea
DA17   32    Male      Driver    Smith
```

[93]:
```
# help(df.sort_index)
```

[94]:
```
df.shape
```

[94]:
```
(5, 4)
```

[95]:
```
df.dtypes
```

[95]:
```
Field
Name      object
Gender    object
Age       object
Job       object
dtype: object
```

[98]:
```
df["Age"] = df["Age"].astype(dtype=np.int64)
```

[99]:
```
df.dtypes
```

[99]:
```
Field
Name      object
Gender    object
Age        int64
Job       object
dtype: object
```

[104]:
```
df["Age"]=df["Age"].astype(dtype=object)
df.dtypes
```

[104]:
```
Field
Name      object
Gender    object
Age       object
Job       object
dtype: object
```

[105]:
```
df["Age"]=pd.to_numeric(arg=df["Age"])
df.dtypes
```

[105]:
```
Field
Name      object
Gender    object
Age        int64
Job       object
dtype: object
```

[107]:
```
df["Gender"]=pd.Categorical(values=df["Gender"])
df.dtypes
```

[107]:
```
Field
Name        object
Gender    category
Age          int64
Job         object
dtype: object
```

[112]:
```
df.select_dtypes(include="object").columns
```

[112]:
```
Index(['Name', 'Job'], dtype='object', name='Field')
```

[113]:
```
for col in df.select_dtypes(include="object").columns:
    df[col]=pd.Categorical(values=df[col])
```

[114]:
```
df.dtypes
```

[114]:
```
Field
Name      category
```

```
Gender    category
Age          int64
Job       category
dtype: object
```

[116]:
```
df.select_dtypes(include=["number", "category"])
```

[116]:
```
Field     Name  Gender  Age         Job
ID
DA03   Michael    Male   35    Lecturer
DA06      Lucy  Female   25  Accountant
DA17     Smith    Male   32      Driver
DA12    Andrea  Female   22    Engineer
DA09      Jane  Female   20    Designer
```

[120]:
```
df_new = df.copy()
```

[121]:
```
df_new
```

[121]:
```
Field     Name  Gender Age         Job
ID
DA03   Michael    Male  35    Lecturer
DA06      Lucy  Female  25  Accountant
DA17     Smith    Male  32      Driver
DA12    Andrea  Female  22    Engineer
DA09      Jane  Female  20    Designer
```

[122]:
```
def encode_gender(gender: str)->int:
    if gender=="Female":
        encode=0
    else:
        encode=1
    return encode
```

[125]:
```
df_new["Gender"]=df["Gender"].apply(func=encode_gender)
df_new
```

[125]:
```
Field     Name  Gender Age         Job
ID
DA03   Michael       1  35    Lecturer
DA06      Lucy       0  25  Accountant
DA17     Smith       1  32      Driver
DA12    Andrea       0  22    Engineer
DA09      Jane       0  20    Designer
```

[128]:
```
new_gender_features = pd.get_dummies(data=df["Gender"]).astype(dtype=int)
new_gender_features
```

[128]:
```
      Female  Male
ID
DA03       0     1
DA06       1     0
DA17       0     1
DA12       1     0
DA09       1     0
```

[129]:
```
df_new = pd.concat(objs=[df_new,new_gender_features],axis=1)
df_new
```

[129]:
```
         Name  Gender Age         Job  Female  Male
ID
DA03  Michael       1  35    Lecturer       0     1
DA06     Lucy       0  25  Accountant       1     0
DA17    Smith       1  32      Driver       0     1
DA12   Andrea       0  22    Engineer       1     0
DA09     Jane       0  20    Designer       1     0
```

[130]:
```
pd.get_dummies(data=df)
```

[130]:
```
      Name_Andrea  Name_Jane  Name_Lucy  Name_Michael  Name_Smith  \
ID
DA03        False      False      False          True       False
DA06        False      False       True         False       False
DA17        False      False      False         False        True
DA12         True      False      False         False       False
DA09        False       True      False         False       False

      Gender_Female  Gender_Male  Age_20  Age_22  Age_25  Age_32  Age_35  \
ID
DA03          False         True   False   False   False   False    True
DA06           True        False   False   False    True   False   False
DA17          False         True   False   False   False    True   False
DA12           True        False   False    True   False   False   False
DA09           True        False    True   False   False   False   False

      Job_Accountant  Job_Designer  Job_Driver  Job_Engineer  Job_Lecturer
ID
DA03           False         False       False         False          True
DA06            True         False       False         False         False
DA17           False         False        True         False         False
DA12           False         False       False          True         False
DA09           False          True       False         False         False
```

[131]:
```
df
```

[131]:
```
Field     Name  Gender Age         Job
ID
DA03   Michael    Male  35    Lecturer
DA06      Lucy  Female  25  Accountant
DA17     Smith    Male  32      Driver
DA12    Andrea  Female  22    Engineer
DA09      Jane  Female  20    Designer
```

[132]:
```
df2 = df.copy()
df2["Age"]=df["Age"].apply(func=lambda x: str(x)+" years old")
df2
```

[132]:
```
Field     Name  Gender           Age         Job
ID
DA03   Michael    Male  35 years old    Lecturer
DA06      Lucy  Female  25 years old  Accountant
DA17     Smith    Male  32 years old      Driver
DA12    Andrea  Female  22 years old    Engineer
DA09      Jane  Female  20 years old    Designer
```

[135]:
```
s = "25 years old"
int(s.split(sep=" ")[0])
```

[135]:
```
25
```

[137]:
```
def extract_age(s:str)->int:
    return int(s.split(sep=" ")[0])
```

[139]:
```
# df2["Age"]=df2["Age"].apply(func=extract_age)
```

[136]:
```
df2["Age"]=df2["Age"].apply(func=lambda s: int(s.split(sep=" ")[0]))
df2
```

[136]:
```
Field     Name  Gender  Age         Job
ID
DA03   Michael    Male   35    Lecturer
DA06      Lucy  Female   25  Accountant
DA17     Smith    Male   32      Driver
DA12    Andrea  Female   22    Engineer
DA09      Jane  Female   20    Designer
```

1.7 Aggregation

[140]:
```
df = pd.DataFrame(
    index=pd.Index(
        name="Id",
        data=["DA03", "DA06", "DA17", "DA12", "DA09", "DA15", "DA01", "DA02"],
    ),
```

```
    data={
        "Name": [
            "Michael",
            "Lucy",
            "Smith",
            "Andrea",
            "Jane",
            "Peter",
            "John",
            "Rebeca",
        ],
        "Gender": [
            "Male",
            "Female",
            "Male",
            "Female",
            "Female",
            "Male",
            "Male",
            "Female",
        ],
        "Age": [31, 25, np.nan, 23, 24, 26, 27, 26],
        "Job": [
            "Lecturer",
            "Accountant",
            "Driver",
            "Engineer",
            "Designer",
            "Scientist",
            "Dentist",
            "Nurse",
        ],
        "Degree": [
            "Master",
            "Doctoral",
            "Bachelor",
            np.nan,
            "Master",
            np.nan,
            "Bachelor",
            "Bachelor",
        ],
        "Email": [
            "da03@domain.com",
            "da06@domain.com",
            "da17@domain.com",
            "da12@domain.com",
```

```
            "da09@domain.com",
            "da15@domain.com",
            np.nan,
            np.nan,
        ],
    },
)
print(df)
```

```
         Name  Gender   Age         Job    Degree            Email
Id
DA03  Michael    Male  31.0    Lecturer    Master  da03@domain.com
DA06     Lucy  Female  25.0  Accountant  Doctoral  da06@domain.com
DA17    Smith    Male   NaN      Driver  Bachelor  da17@domain.com
DA12   Andrea  Female  23.0    Engineer       NaN  da12@domain.com
DA09     Jane  Female  24.0    Designer    Master  da09@domain.com
DA15    Peter    Male  26.0   Scientist       NaN  da15@domain.com
DA01     John    Male  27.0     Dentist  Bachelor              NaN
DA02   Rebeca  Female  26.0       Nurse  Bachelor              NaN
```

[143]:
```
# dir(df)
```

[145]:
```
# help(df.agg)
```

[147]:
```
df.agg(func="count")
```

[147]:
```
Name      8
Gender    8
Age       7
Job       8
Degree    6
Email     6
dtype: int64
```

[149]:
```
df.count()
```

[149]:
```
Name      8
Gender    8
Age       7
Job       8
Degree    6
Email     6
dtype: int64
```

[151]:
```
df.sum(numeric_only=True)
```

[151]:
```
Age    182.0
dtype: float64
```

[154]:
```
df.select_dtypes(include="number").sum()
```

[154]:
```
Age    182.0
dtype: float64
```

[152]:
```
df.agg(func="sum", numeric_only=True)
```

[152]:
```
Age    182.0
dtype: float64
```

[153]:
```
help(df.agg)
```

```
Help on method aggregate in module pandas.core.frame:

aggregate(func=None, axis: 'Axis' = 0, *args, **kwargs) method of
pandas.core.frame.DataFrame instance
    Aggregate using one or more operations over the specified axis.

    Parameters
    ----------
    func : function, str, list or dict
        Function to use for aggregating the data. If a function, must either
        work when passed a DataFrame or when passed to DataFrame.apply.

        Accepted combinations are:

        - function
        - string function name
        - list of functions and/or function names, e.g. ``[np.sum, 'mean']``
        - dict of axis labels -> functions, function names or list of such.
    axis : {0 or 'index', 1 or 'columns'}, default 0
            If 0 or 'index': apply function to each column.
            If 1 or 'columns': apply function to each row.
    *args
        Positional arguments to pass to `func`.
    **kwargs
        Keyword arguments to pass to `func`.

    Returns
    -------
    scalar, Series or DataFrame

        The return can be:

        * scalar : when Series.agg is called with single function
        * Series : when DataFrame.agg is called with a single function
        * DataFrame : when DataFrame.agg is called with several functions
```

```
        Return scalar, Series or DataFrame.

    The aggregation operations are always performed over an axis, either the
    index (default) or the column axis. This behavior is different from
    `numpy` aggregation functions (`mean`, `median`, `prod`, `sum`, `std`,
    `var`), where the default is to compute the aggregation of the flattened
    array, e.g., ``numpy.mean(arr_2d)`` as opposed to
    ``numpy.mean(arr_2d, axis=0)``.

    `agg` is an alias for `aggregate`. Use the alias.

    See Also
    --------
    DataFrame.apply : Perform any type of operations.
    DataFrame.transform : Perform transformation type operations.
    core.groupby.GroupBy : Perform operations over groups.
    core.resample.Resampler : Perform operations over resampled bins.
    core.window.Rolling : Perform operations over rolling window.
    core.window.Expanding : Perform operations over expanding window.
    core.window.ExponentialMovingWindow : Perform operation over exponential
weighted
        window.

    Notes
    -----
    `agg` is an alias for `aggregate`. Use the alias.

    Functions that mutate the passed object can produce unexpected
    behavior or errors and are not supported. See :ref:`gotchas.udf-mutation`
    for more details.

    A passed user-defined-function will be passed a Series for evaluation.

    Examples
    --------
    >>> df = pd.DataFrame([[1, 2, 3],
    ...                    [4, 5, 6],
    ...                    [7, 8, 9],
    ...                    [np.nan, np.nan, np.nan]],
    ...                   columns=['A', 'B', 'C'])

    Aggregate these functions over the rows.

    >>> df.agg(['sum', 'min'])
            A     B     C
    sum  12.0  15.0  18.0
    min   1.0   2.0   3.0
```

```
    Different aggregations per column.

    >>> df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
            A    B
    sum  12.0  NaN
    min   1.0  2.0
    max   NaN  8.0

    Aggregate different functions over the columns and rename the index of the
resulting
    DataFrame.

    >>> df.agg(x=('A', max), y=('B', 'min'), z=('C', np.mean))
         A    B    C
    x  7.0  NaN  NaN
    y  NaN  2.0  NaN
    z  NaN  NaN  6.0

    Aggregate over the columns.

    >>> df.agg("mean", axis="columns")
    0    2.0
    1    5.0
    2    8.0
    3    NaN
    dtype: float64
```

[157]:
```
# help(df.mean)
```

[161]:
```
df[["Age"]].agg(func=["count", "sum", "mean"])
```

[161]:

	Age
count	7.0
sum	182.0
mean	26.0

1.8 Groupby Method

[162]:
```
df = pd.DataFrame(
    index=pd.Index(
        name="Id",
        data=["DA03", "DA06", "DA17", "DA12", "DA09", "DA15", "DA01", "DA02"],
    ),
    data={
        "Name": [
            "Michael",
            "Lucy",
```

```
            "Smith",
            "Andrea",
            "Jane",
            "Peter",
            "John",
            "Rebeca",
        ],
        "Gender": [
            "Male",
            "Female",
            "Male",
            "Female",
            "Female",
            "Male",
            "Male",
            "Female",
        ],
        "Age": [31, 25, np.nan, 23, 24, 26, 27, 26],
        "Job": [
            "Lecturer",
            "Accountant",
            "Driver",
            "Engineer",
            "Designer",
            "Scientist",
            "Dentist",
            "Nurse",
        ],
        "Degree": [
            "Master",
            "Doctoral",
            "Bachelor",
            np.nan,
            "Master",
            np.nan,
            "Bachelor",
            "Bachelor",
        ],
        "Email": [
            "da03@domain.com",
            "da06@domain.com",
            "da17@domain.com",
            "da12@domain.com",
            "da09@domain.com",
            "da15@domain.com",
            np.nan,
            np.nan,
```

```
        ],
    },
)
print(df)
```

```
         Name  Gender   Age         Job    Degree             Email
Id
DA03  Michael    Male  31.0    Lecturer    Master  da03@domain.com
DA06     Lucy  Female  25.0  Accountant  Doctoral  da06@domain.com
DA17    Smith    Male   NaN      Driver  Bachelor  da17@domain.com
DA12   Andrea  Female  23.0    Engineer       NaN  da12@domain.com
DA09     Jane  Female  24.0    Designer    Master  da09@domain.com
DA15    Peter    Male  26.0   Scientist       NaN  da15@domain.com
DA01     John    Male  27.0     Dentist  Bachelor              NaN
DA02   Rebeca  Female  26.0       Nurse  Bachelor              NaN
```

[164]:
```
gender_gp = df.groupby(by="Gender")
```

[165]:
```
dir(gender_gp)
```

[165]:
```
['Age',
 'Degree',
 'Email',
 'Gender',
 'Job',
 'Name',
 '_DataFrameGroupBy__examples_dataframe_doc',
 '__annotations__',
 '__class__',
 '__class_getitem__',
 '__delattr__',
 '__dict__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattr__',
 '__getattribute__',
 '__getitem__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
```

```
'__lt__',
'__module__',
'__ne__',
'__new__',
'__orig_bases__',
'__parameters__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__slots__',
'__str__',
'__subclasshook__',
'__weakref__',
'_accessors',
'_agg_examples_doc',
'_agg_general',
'_agg_py_fallback',
'_aggregate_frame',
'_aggregate_with_numba',
'_apply_filter',
'_apply_to_column_groupbys',
'_ascending_count',
'_bool_agg',
'_choose_path',
'_concat_objects',
'_constructor',
'_cumcount_array',
'_cython_agg_general',
'_cython_transform',
'_define_paths',
'_descending_count',
'_dir_additions',
'_dir_deletions',
'_fill',
'_get_cythonized_result',
'_get_data_to_aggregate',
'_get_index',
'_get_indices',
'_gotitem',
'_hidden_attrs',
'_indexed_output_to_ndframe',
'_insert_inaxis_grouper',
'_internal_names',
'_internal_names_set',
'_is_protocol',
```

```
'_iterate_column_groupbys',
'_iterate_slices',
'_make_mask_from_int',
'_make_mask_from_list',
'_make_mask_from_positional_indexer',
'_make_mask_from_slice',
'_make_mask_from_tuple',
'_mask_selected_obj',
'_maybe_transpose_result',
'_nth',
'_numba_agg_general',
'_numba_prep',
'_obj_1d_constructor',
'_obj_with_exclusions',
'_op_via_apply',
'_positional_selector',
'_python_agg_general',
'_python_apply_general',
'_reindex_output',
'_reset_cache',
'_selected_obj',
'_selection',
'_selection_list',
'_set_result_index_ordered',
'_transform',
'_transform_general',
'_transform_with_numba',
'_value_counts',
'_wrap_agged_manager',
'_wrap_aggregated_output',
'_wrap_applied_output',
'_wrap_applied_output_series',
'_wrap_transform_fast_result',
'agg',
'aggregate',
'all',
'any',
'apply',
'bfill',
'boxplot',
'corr',
'corrwith',
'count',
'cov',
'cumcount',
'cummax',
'cummin',
```

```
'cumprod',
'cumsum',
'describe',
'diff',
'dtypes',
'ewm',
'expanding',
'ffill',
'fillna',
'filter',
'first',
'get_group',
'groups',
'head',
'hist',
'idxmax',
'idxmin',
'indices',
'last',
'max',
'mean',
'median',
'min',
'ndim',
'ngroup',
'ngroups',
'nth',
'nunique',
'ohlc',
'pct_change',
'pipe',
'plot',
'prod',
'quantile',
'rank',
'resample',
'rolling',
'sample',
'sem',
'shift',
'size',
'skew',
'std',
'sum',
'tail',
'take',
'transform',
```

```
 'value_counts',
 'var']
```

[166]:
```
gender_gp.get_group(name="Male")
```

[166]:
```
         Name Gender   Age        Job    Degree            Email
Id
DA03  Michael   Male  31.0   Lecturer    Master  da03@domain.com
DA17    Smith   Male   NaN     Driver  Bachelor  da17@domain.com
DA15    Peter   Male  26.0  Scientist       NaN  da15@domain.com
DA01     John   Male  27.0    Dentist  Bachelor              NaN
```

[169]:
```
gender_gp.get_group(name="Female")
```

[169]:
```
        Name  Gender   Age         Job    Degree            Email
Id
DA06    Lucy  Female  25.0  Accountant  Doctoral  da06@domain.com
DA12  Andrea  Female  23.0    Engineer       NaN  da12@domain.com
DA09    Jane  Female  24.0    Designer    Master  da09@domain.com
DA02  Rebeca  Female  26.0       Nurse  Bachelor              NaN
```

[168]:
```
gender_gp.agg(func="mean", numeric_only=True)
```

[168]:
```
          Age
Gender
Female  24.5
Male    28.0
```

[170]:
```
gender_gp.groups
```

[170]:
```
{'Female': ['DA06', 'DA12', 'DA09', 'DA02'], 'Male': ['DA03', 'DA17', 'DA15',
'DA01']}
```

[171]:
```
df.loc[['DA06', 'DA12', 'DA09', 'DA02']]
```

[171]:
```
        Name  Gender   Age         Job    Degree            Email
Id
DA06    Lucy  Female  25.0  Accountant  Doctoral  da06@domain.com
DA12  Andrea  Female  23.0    Engineer       NaN  da12@domain.com
DA09    Jane  Female  24.0    Designer    Master  da09@domain.com
DA02  Rebeca  Female  26.0       Nurse  Bachelor              NaN
```

[172]:
```
gender_gp.mean(numeric_only=True)
```

[172]:
```
          Age
Gender
Female  24.5
Male    28.0
```

1.9 PivotTable

[173]:
```
df = pd.DataFrame(
    index=pd.Index(
        name="Id",
        data=["DA03", "DA06", "DA17", "DA12", "DA09", "DA15", "DA01", "DA02"],
    ),
    data={
        "Name": [
            "Michael",
            "Lucy",
            "Smith",
            "Andrea",
            "Jane",
            "Peter",
            "John",
            "Rebeca",
        ],
        "Gender": [
            "Male",
            "Female",
            "Male",
            "Female",
            "Female",
            "Male",
            "Male",
            "Female",
        ],
        "Age": [31, 25, np.nan, 23, 24, 26, 27, 26],
        "Job": [
            "Lecturer",
            "Accountant",
            "Driver",
            "Engineer",
            "Designer",
            "Scientist",
            "Dentist",
            "Nurse",
        ],
        "Degree": [
            "Master",
            "Doctoral",
            "Bachelor",
            np.nan,
            "Master",
            np.nan,
            "Bachelor",
            "Bachelor",
```

```
        ],
        "Email": [
            "da03@domain.com",
            "da06@domain.com",
            "da17@domain.com",
            "da12@domain.com",
            "da09@domain.com",
            "da15@domain.com",
            np.nan,
            np.nan,
        ],
    },
)
print(df)
```

```
         Name  Gender   Age         Job    Degree            Email
Id
DA03  Michael    Male  31.0    Lecturer    Master  da03@domain.com
DA06     Lucy  Female  25.0  Accountant  Doctoral  da06@domain.com
DA17    Smith    Male   NaN      Driver  Bachelor  da17@domain.com
DA12   Andrea  Female  23.0    Engineer       NaN  da12@domain.com
DA09     Jane  Female  24.0    Designer    Master  da09@domain.com
DA15    Peter    Male  26.0   Scientist       NaN  da15@domain.com
DA01     John    Male  27.0     Dentist  Bachelor              NaN
DA02   Rebeca  Female  26.0       Nurse  Bachelor              NaN
```

[174]:
```
df.pivot_table(
    index="Gender",
    columns="Degree",
    values="Age",
    aggfunc="mean",
)
```

[174]:
```
Degree  Bachelor  Doctoral  Master
Gender
Female      26.0      25.0    24.0
Male        27.0       NaN    31.0
```

[177]:
```
df.groupby(by=["Gender","Degree"]).count()
```

[177]:
```
                 Name  Age  Job  Email
Gender Degree
Female Bachelor     1    1    1      0
       Doctoral     1    1    1      1
       Master       1    1    1      1
Male   Bachelor     2    1    2      1
       Master       1    1    1      1
```

[178]:
```
df.groupby(by=["Gender","Degree"]).get_group(name=("Male","Bachelor"))
```

[178]:

Id	Name	Gender	Age	Job	Degree	Email
DA17	Smith	Male	NaN	Driver	Bachelor	da17@domain.com
DA01	John	Male	27.0	Dentist	Bachelor	NaN

[175]:
```
df.groupby(by=["Gender","Degree"]).mean(numeric_only=True)
```

[175]:

Gender	Degree	Age
Female	Bachelor	26.0
	Doctoral	25.0
	Master	24.0
Male	Bachelor	27.0
	Master	31.0

[179]:
```
import seaborn as sns
```

[180]:
```
p = sns.load_dataset("penguins")
p
```

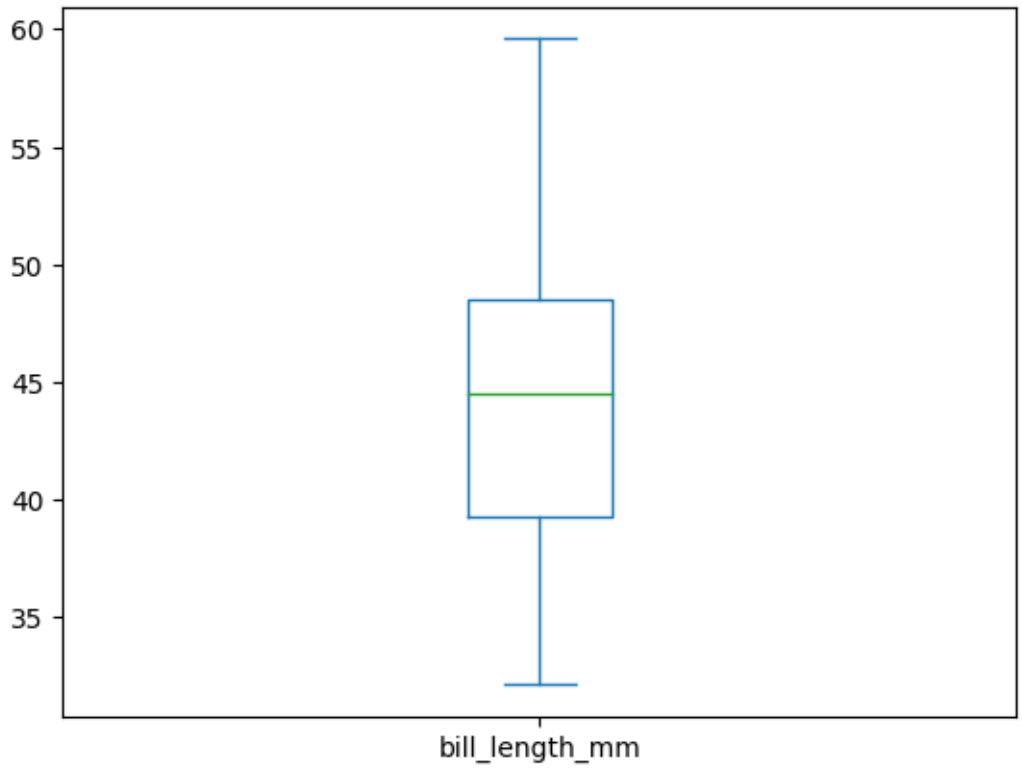
[180]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
..	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

```
[344 rows x 7 columns]
```

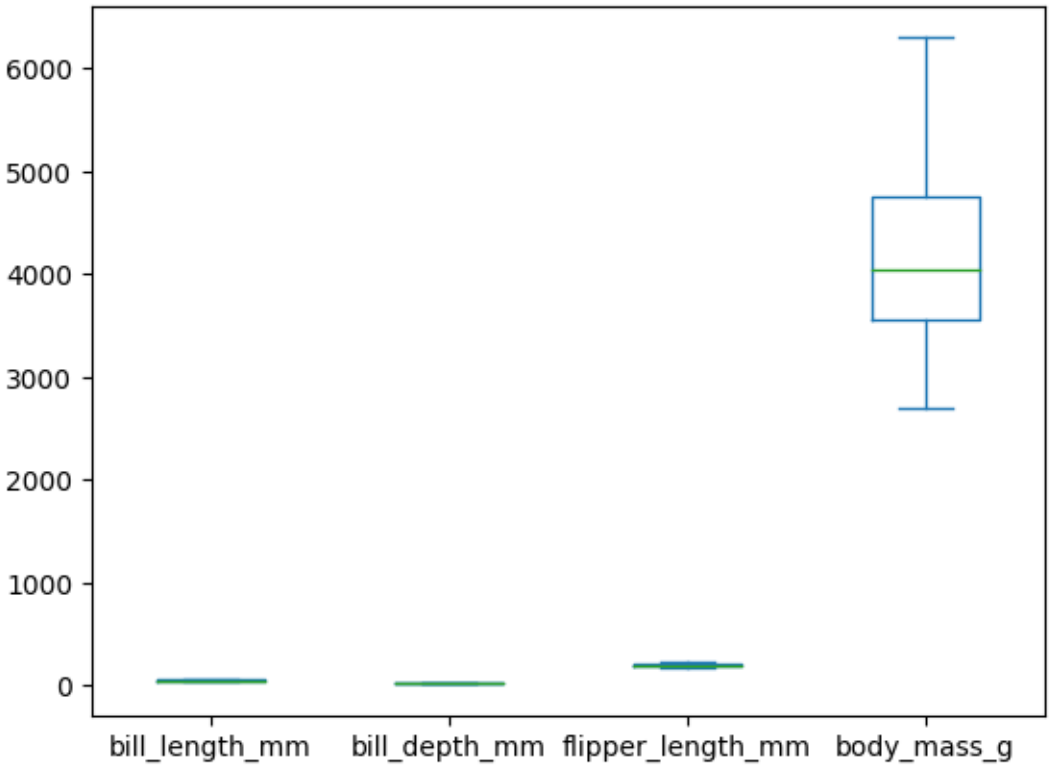
[181]:
```
p["bill_length_mm"].plot(kind="box")
```

[181]: <Axes: >



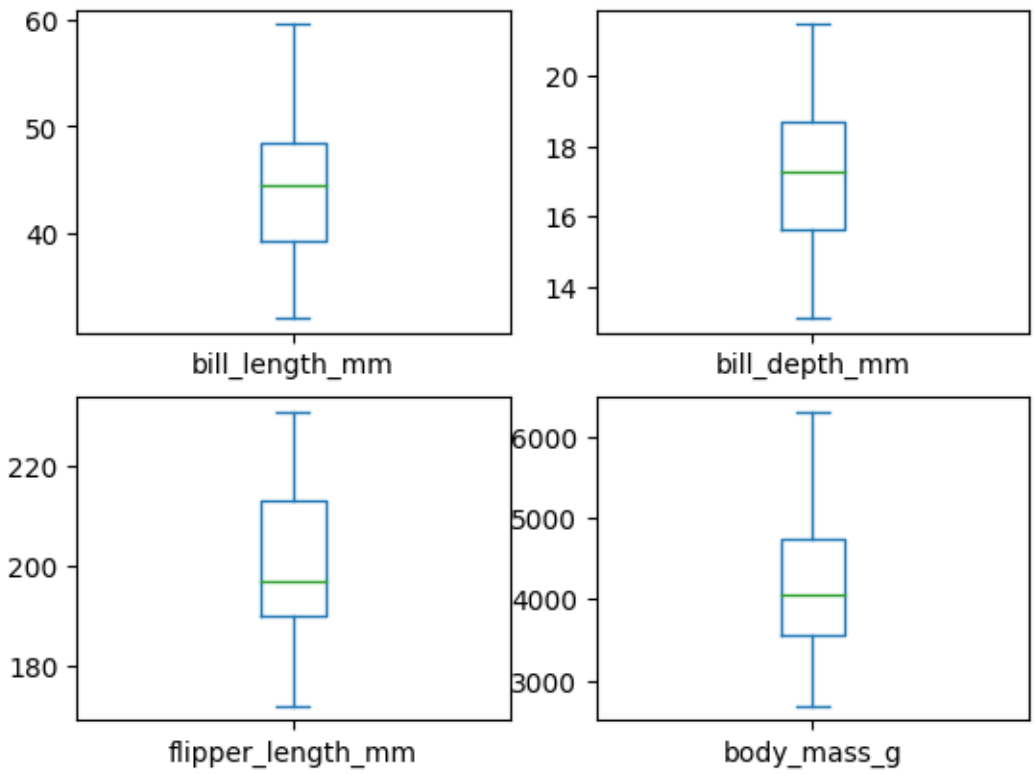
[182]:
```
p.select_dtypes(include="number").plot(kind="box")
```

[182]: <Axes: >



[184]:
```
p.select_dtypes(include="number").plot(
    kind="box",
    subplots=True,
    layout=(2, 2),
)
```

[184]:
```
bill_length_mm          Axes(0.125,0.53;0.352273x0.35)
bill_depth_mm        Axes(0.547727,0.53;0.352273x0.35)
flipper_length_mm       Axes(0.125,0.11;0.352273x0.35)
body_mass_g          Axes(0.547727,0.11;0.352273x0.35)
dtype: object
```



[185]:
```
help(pd.DataFrame.plot)
```

```
Help on class PlotAccessor in module pandas.plotting._core:

class PlotAccessor(pandas.core.base.PandasObject)
 |  PlotAccessor(data) -> 'None'
 |
 |  Make plots of Series or DataFrame.
 |
 |  Uses the backend specified by the
 |  option ``plotting.backend``. By default, matplotlib is used.
 |
 |  Parameters
 |  ----------
 |  data : Series or DataFrame
 |      The object for which the method is called.
 |  x : label or position, default None
 |      Only used if data is a DataFrame.
 |  y : label, position or list of label, positions, default None
 |      Allows plotting of one column versus another. Only used if data is a
 |      DataFrame.
 |  kind : str
```

```
 |      The kind of plot to produce:
 |
 |      - 'line' : line plot (default)
 |      - 'bar' : vertical bar plot
 |      - 'barh' : horizontal bar plot
 |      - 'hist' : histogram
 |      - 'box' : boxplot
 |      - 'kde' : Kernel Density Estimation plot
 |      - 'density' : same as 'kde'
 |      - 'area' : area plot
 |      - 'pie' : pie plot
 |      - 'scatter' : scatter plot (DataFrame only)
 |      - 'hexbin' : hexbin plot (DataFrame only)
 |  ax : matplotlib axes object, default None
 |      An axes of the current figure.
 |  subplots : bool or sequence of iterables, default False
 |      Whether to group columns into subplots:
 |
 |      - ``False`` : No subplots will be used
 |      - ``True`` : Make separate subplots for each column.
 |      - sequence of iterables of column labels: Create a subplot for each
 |        group of columns. For example `[('a', 'c'), ('b', 'd')]` will
 |        create 2 subplots: one with columns 'a' and 'c', and one
 |        with columns 'b' and 'd'. Remaining columns that aren't specified
 |        will be plotted in additional subplots (one per column).
 |
 |        .. versionadded:: 1.5.0
 |
 |  sharex : bool, default True if ax is None else False
 |      In case ``subplots=True``, share x axis and set some x axis labels
 |      to invisible; defaults to True if ax is None otherwise False if
 |      an ax is passed in; Be aware, that passing in both an ax and
 |      ``sharex=True`` will alter all x axis labels for all axis in a figure.
 |  sharey : bool, default False
 |      In case ``subplots=True``, share y axis and set some y axis labels to
invisible.
 |  layout : tuple, optional
 |      (rows, columns) for the layout of subplots.
 |  figsize : a tuple (width, height) in inches
 |      Size of a figure object.
 |  use_index : bool, default True
 |      Use index as ticks for x axis.
 |  title : str or list
 |      Title to use for the plot. If a string is passed, print the string
 |      at the top of the figure. If a list is passed and `subplots` is
 |      True, print each item in the list above the corresponding subplot.
 |  grid : bool, default None (matlab style default)
 |      Axis grid lines.
```

```
 |  legend : bool or {'reverse'}
 |      Place legend on axis subplots.
 |  style : list or dict
 |      The matplotlib line style per column.
 |  logx : bool or 'sym', default False
 |      Use log scaling or symlog scaling on x axis.
 |
 |  logy : bool or 'sym' default False
 |      Use log scaling or symlog scaling on y axis.
 |
 |  loglog : bool or 'sym', default False
 |      Use log scaling or symlog scaling on both x and y axes.
 |
 |  xticks : sequence
 |      Values to use for the xticks.
 |  yticks : sequence
 |      Values to use for the yticks.
 |  xlim : 2-tuple/list
 |      Set the x limits of the current axes.
 |  ylim : 2-tuple/list
 |      Set the y limits of the current axes.
 |  xlabel : label, optional
 |      Name to use for the xlabel on x-axis. Default uses index name as xlabel,
or the
 |      x-column name for planar plots.
 |
 |      .. versionadded:: 1.1.0
 |
 |      .. versionchanged:: 1.2.0
 |
 |         Now applicable to planar plots (`scatter`, `hexbin`).
 |
 |      .. versionchanged:: 2.0.0
 |
 |          Now applicable to histograms.
 |
 |  ylabel : label, optional
 |      Name to use for the ylabel on y-axis. Default will show no ylabel, or
the
 |      y-column name for planar plots.
 |
 |      .. versionadded:: 1.1.0
 |
 |      .. versionchanged:: 1.2.0
 |
 |         Now applicable to planar plots (`scatter`, `hexbin`).
 |
 |      .. versionchanged:: 2.0.0
```

```
|
|         Now applicable to histograms.
|
|  rot : float, default None
|      Rotation for ticks (xticks for vertical, yticks for horizontal
|      plots).
|  fontsize : float, default None
|      Font size for xticks and yticks.
|  colormap : str or matplotlib colormap object, default None
|      Colormap to select colors from. If string, load colormap with that
|      name from matplotlib.
|  colorbar : bool, optional
|      If True, plot colorbar (only relevant for 'scatter' and 'hexbin'
|      plots).
|  position : float
|      Specify relative alignments for bar plot layout.
|      From 0 (left/bottom-end) to 1 (right/top-end). Default is 0.5
|      (center).
|  table : bool, Series or DataFrame, default False
|      If True, draw a table using the data in the DataFrame and the data
|      will be transposed to meet matplotlib's default layout.
|      If a Series or DataFrame is passed, use passed data to draw a
|      table.
|  yerr : DataFrame, Series, array-like, dict and str
|      See :ref:`Plotting with Error Bars <visualization.errorbars>` for
|      detail.
|  xerr : DataFrame, Series, array-like, dict and str
|      Equivalent to yerr.
|  stacked : bool, default False in line and bar plots, and True in area plot
|      If True, create stacked plot.
|  secondary_y : bool or sequence, default False
|      Whether to plot on the secondary y-axis if a list/tuple, which
|      columns to plot on secondary y-axis.
|  mark_right : bool, default True
|      When using a secondary_y axis, automatically mark the column
|      labels with "(right)" in the legend.
|  include_bool : bool, default is False
|      If True, boolean values can be plotted.
|  backend : str, default None
|      Backend to use instead of the backend specified in the option
|      ``plotting.backend``. For instance, 'matplotlib'. Alternatively, to
|      specify the ``plotting.backend`` for the whole session, set
|      ``pd.options.plotting.backend``.
|  **kwargs
|      Options to pass to matplotlib plotting method.
|
|  Returns
|  -------
```

```
|  :class:`matplotlib.axes.Axes` or numpy.ndarray of them
|      If the backend is not the default matplotlib one, the return value
|      will be the object returned by the backend.
|
|  Notes
|  -----
|  - See matplotlib documentation online for more on this subject
|  - If `kind` = 'bar' or 'barh', you can specify relative alignments
|    for bar plot layout by `position` keyword.
|    From 0 (left/bottom-end) to 1 (right/top-end). Default is 0.5
|    (center)
|
|  Method resolution order:
|      PlotAccessor
|      pandas.core.base.PandasObject
|      pandas.core.accessor.DirNamesMixin
|      builtins.object
|
|  Methods defined here:
|
|  __call__(self, *args, **kwargs)
|      Make plots of Series or DataFrame.
|
|      Uses the backend specified by the
|      option ``plotting.backend``. By default, matplotlib is used.
|
|      Parameters
|      ----------
|      data : Series or DataFrame
|          The object for which the method is called.
|      x : label or position, default None
|          Only used if data is a DataFrame.
|      y : label, position or list of label, positions, default None
|          Allows plotting of one column versus another. Only used if data is a
|          DataFrame.
|      kind : str
|          The kind of plot to produce:
|
|          - 'line' : line plot (default)
|          - 'bar' : vertical bar plot
|          - 'barh' : horizontal bar plot
|          - 'hist' : histogram
|          - 'box' : boxplot
|          - 'kde' : Kernel Density Estimation plot
|          - 'density' : same as 'kde'
|          - 'area' : area plot
|          - 'pie' : pie plot
|          - 'scatter' : scatter plot (DataFrame only)
```

```
 |          - 'hexbin' : hexbin plot (DataFrame only)
 |      ax : matplotlib axes object, default None
 |          An axes of the current figure.
 |      subplots : bool or sequence of iterables, default False
 |          Whether to group columns into subplots:
 |
 |          - ``False`` : No subplots will be used
 |          - ``True`` : Make separate subplots for each column.
 |          - sequence of iterables of column labels: Create a subplot for each
 |            group of columns. For example `[('a', 'c'), ('b', 'd')]` will
 |            create 2 subplots: one with columns 'a' and 'c', and one
 |            with columns 'b' and 'd'. Remaining columns that aren't specified
 |            will be plotted in additional subplots (one per column).
 |
 |            .. versionadded:: 1.5.0
 |
 |      sharex : bool, default True if ax is None else False
 |          In case ``subplots=True``, share x axis and set some x axis labels
 |          to invisible; defaults to True if ax is None otherwise False if
 |          an ax is passed in; Be aware, that passing in both an ax and
 |          ``sharex=True`` will alter all x axis labels for all axis in a
figure.
 |      sharey : bool, default False
 |          In case ``subplots=True``, share y axis and set some y axis labels
to invisible.
 |      layout : tuple, optional
 |          (rows, columns) for the layout of subplots.
 |      figsize : a tuple (width, height) in inches
 |          Size of a figure object.
 |      use_index : bool, default True
 |          Use index as ticks for x axis.
 |      title : str or list
 |          Title to use for the plot. If a string is passed, print the string
 |          at the top of the figure. If a list is passed and `subplots` is
 |          True, print each item in the list above the corresponding subplot.
 |      grid : bool, default None (matlab style default)
 |          Axis grid lines.
 |      legend : bool or {'reverse'}
 |          Place legend on axis subplots.
 |      style : list or dict
 |          The matplotlib line style per column.
 |      logx : bool or 'sym', default False
 |          Use log scaling or symlog scaling on x axis.
 |
 |      logy : bool or 'sym' default False
 |          Use log scaling or symlog scaling on y axis.
 |
 |      loglog : bool or 'sym', default False
```

```
 |          Use log scaling or symlog scaling on both x and y axes.
 |
 |      xticks : sequence
 |          Values to use for the xticks.
 |      yticks : sequence
 |          Values to use for the yticks.
 |      xlim : 2-tuple/list
 |          Set the x limits of the current axes.
 |      ylim : 2-tuple/list
 |          Set the y limits of the current axes.
 |      xlabel : label, optional
 |          Name to use for the xlabel on x-axis. Default uses index name as
xlabel, or the
 |          x-column name for planar plots.
 |
 |          .. versionadded:: 1.1.0
 |
 |          .. versionchanged:: 1.2.0
 |
 |             Now applicable to planar plots (`scatter`, `hexbin`).
 |
 |          .. versionchanged:: 2.0.0
 |
 |              Now applicable to histograms.
 |
 |      ylabel : label, optional
 |          Name to use for the ylabel on y-axis. Default will show no ylabel,
or the
 |          y-column name for planar plots.
 |
 |          .. versionadded:: 1.1.0
 |
 |          .. versionchanged:: 1.2.0
 |
 |             Now applicable to planar plots (`scatter`, `hexbin`).
 |
 |          .. versionchanged:: 2.0.0
 |
 |              Now applicable to histograms.
 |
 |      rot : float, default None
 |          Rotation for ticks (xticks for vertical, yticks for horizontal
 |          plots).
 |      fontsize : float, default None
 |          Font size for xticks and yticks.
 |      colormap : str or matplotlib colormap object, default None
 |          Colormap to select colors from. If string, load colormap with that
 |          name from matplotlib.
```

```
 |      colorbar : bool, optional
 |          If True, plot colorbar (only relevant for 'scatter' and 'hexbin'
 |          plots).
 |      position : float
 |          Specify relative alignments for bar plot layout.
 |          From 0 (left/bottom-end) to 1 (right/top-end). Default is 0.5
 |          (center).
 |      table : bool, Series or DataFrame, default False
 |          If True, draw a table using the data in the DataFrame and the data
 |          will be transposed to meet matplotlib's default layout.
 |          If a Series or DataFrame is passed, use passed data to draw a
 |          table.
 |      yerr : DataFrame, Series, array-like, dict and str
 |          See :ref:`Plotting with Error Bars <visualization.errorbars>` for
 |          detail.
 |      xerr : DataFrame, Series, array-like, dict and str
 |          Equivalent to yerr.
 |      stacked : bool, default False in line and bar plots, and True in area
plot
 |          If True, create stacked plot.
 |      secondary_y : bool or sequence, default False
 |          Whether to plot on the secondary y-axis if a list/tuple, which
 |          columns to plot on secondary y-axis.
 |      mark_right : bool, default True
 |          When using a secondary_y axis, automatically mark the column
 |          labels with "(right)" in the legend.
 |      include_bool : bool, default is False
 |          If True, boolean values can be plotted.
 |      backend : str, default None
 |          Backend to use instead of the backend specified in the option
 |          ``plotting.backend``. For instance, 'matplotlib'. Alternatively, to
 |          specify the ``plotting.backend`` for the whole session, set
 |          ``pd.options.plotting.backend``.
 |      **kwargs
 |          Options to pass to matplotlib plotting method.
 |
 |      Returns
 |      -------
 |      :class:`matplotlib.axes.Axes` or numpy.ndarray of them
 |          If the backend is not the default matplotlib one, the return value
 |          will be the object returned by the backend.
 |
 |      Notes
 |      -----
 |      - See matplotlib documentation online for more on this subject
 |      - If `kind` = 'bar' or 'barh', you can specify relative alignments
 |        for bar plot layout by `position` keyword.
 |        From 0 (left/bottom-end) to 1 (right/top-end). Default is 0.5
```

```
 |        (center)
 |
 |  __init__(self, data) -> 'None'
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  area(self, x=None, y=None, stacked: 'bool' = True, **kwargs) ->
'PlotAccessor'
 |      Draw a stacked area plot.
 |
 |      An area plot displays quantitative data visually.
 |      This function wraps the matplotlib area function.
 |
 |      Parameters
 |      ----------
 |      x : label or position, optional
 |          Coordinates for the X axis. By default uses the index.
 |      y : label or position, optional
 |          Column to plot. By default uses all columns.
 |      stacked : bool, default True
 |          Area plots are stacked by default. Set to False to create a
 |          unstacked plot.
 |      **kwargs
 |          Additional keyword arguments are documented in
 |          :meth:`DataFrame.plot`.
 |
 |      Returns
 |      -------
 |      matplotlib.axes.Axes or numpy.ndarray
 |          Area plot, or array of area plots if subplots is True.
 |
 |      See Also
 |      --------
 |      DataFrame.plot : Make plots of DataFrame using matplotlib / pylab.
 |
 |      Examples
 |      --------
 |      Draw an area plot based on basic business metrics:
 |
 |      .. plot::
 |          :context: close-figs
 |
 |          >>> df = pd.DataFrame({
 |          ...     'sales': [3, 2, 3, 9, 10, 6],
 |          ...     'signups': [5, 5, 6, 12, 14, 13],
 |          ...     'visits': [20, 42, 28, 62, 81, 50],
 |          ... }, index=pd.date_range(start='2018/01/01', end='2018/07/01',
 |          ...                        freq='M'))
 |          >>> ax = df.plot.area()
```

```
|
|      Area plots are stacked by default. To produce an unstacked plot,
|      pass ``stacked=False``:
|
|      .. plot::
|          :context: close-figs
|
|          >>> ax = df.plot.area(stacked=False)
|
|      Draw an area plot for a single column:
|
|      .. plot::
|          :context: close-figs
|
|          >>> ax = df.plot.area(y='sales')
|
|      Draw with a different `x`:
|
|      .. plot::
|          :context: close-figs
|
|          >>> df = pd.DataFrame({
|          ...     'sales': [3, 2, 3],
|          ...     'visits': [20, 42, 28],
|          ...     'day': [1, 2, 3],
|          ... })
|          >>> ax = df.plot.area(x='day')
|
|  bar(self, x=None, y=None, **kwargs) -> 'PlotAccessor'
|      Vertical bar plot.
|
|      A bar plot is a plot that presents categorical data with
|      rectangular bars with lengths proportional to the values that they
|      represent. A bar plot shows comparisons among discrete categories. One
|      axis of the plot shows the specific categories being compared, and the
|      other axis represents a measured value.
|
|      Parameters
|      ----------
|      x : label or position, optional
|          Allows plotting of one column versus another. If not specified,
|          the index of the DataFrame is used.
|      y : label or position, optional
|          Allows plotting of one column versus another. If not specified,
|          all numerical columns are used.
|      color : str, array-like, or dict, optional
|          The color for each of the DataFrame's columns. Possible values are:
|
```

```
 |          - A single color string referred to by name, RGB or RGBA code,
 |              for instance 'red' or '#a98d19'.
 |
 |          - A sequence of color strings referred to by name, RGB or RGBA
 |              code, which will be used for each column recursively. For
 |              instance ['green','yellow'] each column's bar will be filled in
 |              green or yellow, alternatively. If there is only a single column
to
 |              be plotted, then only the first color from the color list will
be
 |              used.
 |
 |          - A dict of the form {column name : color}, so that each column will
be
 |              colored accordingly. For example, if your columns are called `a`
and
 |              `b`, then passing {'a': 'green', 'b': 'red'} will color bars for
 |              column `a` in green and bars for column `b` in red.
 |
 |          .. versionadded:: 1.1.0
 |
 |      **kwargs
 |          Additional keyword arguments are documented in
 |          :meth:`DataFrame.plot`.
 |
 |      Returns
 |      -------
 |      matplotlib.axes.Axes or np.ndarray of them
 |          An ndarray is returned with one :class:`matplotlib.axes.Axes`
 |          per column when ``subplots=True``.
 |
 |              See Also
 |              --------
 |              DataFrame.plot.barh : Horizontal bar plot.
 |              DataFrame.plot : Make plots of a DataFrame.
 |              matplotlib.pyplot.bar : Make a bar plot with matplotlib.
 |
 |              Examples
 |              --------
 |              Basic plot.
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> df = pd.DataFrame({'lab':['A', 'B', 'C'], 'val':[10, 30,
20]})
 |                  >>> ax = df.plot.bar(x='lab', y='val', rot=0)
 |
```

```
 |              Plot a whole dataframe to a bar plot. Each column is assigned a
 |              distinct color, and each row is nested in a group along the
 |              horizontal axis.
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
 |                  >>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
 |                  >>> index = ['snail', 'pig', 'elephant',
 |                  ...          'rabbit', 'giraffe', 'coyote', 'horse']
 |                  >>> df = pd.DataFrame({'speed': speed,
 |                  ...                     'lifespan': lifespan}, index=index)
 |                  >>> ax = df.plot.bar(rot=0)
 |
 |              Plot stacked bar charts for the DataFrame
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> ax = df.plot.bar(stacked=True)
 |
 |              Instead of nesting, the figure can be split by column with
 |              ``subplots=True``. In this case, a :class:`numpy.ndarray` of
 |              :class:`matplotlib.axes.Axes` are returned.
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> axes = df.plot.bar(rot=0, subplots=True)
 |                  >>> axes[1].legend(loc=2)  # doctest: +SKIP
 |
 |              If you don't like the default colours, you can specify how you'd
 |              like each column to be colored.
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> axes = df.plot.bar(
 |                  ...     rot=0, subplots=True, color={"speed": "red",
"lifespan": "green"}
 |                  ... )
 |                  >>> axes[1].legend(loc=2)  # doctest: +SKIP
 |
 |              Plot a single column.
 |
 |              .. plot::
 |                  :context: close-figs
```

```
 |
 |                 >>> ax = df.plot.bar(y='speed', rot=0)
 |
 |             Plot only selected categories for the DataFrame.
 |
 |             .. plot::
 |                 :context: close-figs
 |
 |                 >>> ax = df.plot.bar(x='lifespan', rot=0)
 |
 |  barh(self, x=None, y=None, **kwargs) -> 'PlotAccessor'
 |      Make a horizontal bar plot.
 |
 |      A horizontal bar plot is a plot that presents quantitative data with
 |      rectangular bars with lengths proportional to the values that they
 |      represent. A bar plot shows comparisons among discrete categories. One
 |      axis of the plot shows the specific categories being compared, and the
 |      other axis represents a measured value.
 |
 |      Parameters
 |      ----------
 |      x : label or position, optional
 |          Allows plotting of one column versus another. If not specified,
 |          the index of the DataFrame is used.
 |      y : label or position, optional
 |          Allows plotting of one column versus another. If not specified,
 |          all numerical columns are used.
 |      color : str, array-like, or dict, optional
 |          The color for each of the DataFrame's columns. Possible values are:
 |
 |          - A single color string referred to by name, RGB or RGBA code,
 |              for instance 'red' or '#a98d19'.
 |
 |          - A sequence of color strings referred to by name, RGB or RGBA
 |              code, which will be used for each column recursively. For
 |              instance ['green','yellow'] each column's bar will be filled in
 |              green or yellow, alternatively. If there is only a single column
to
 |              be plotted, then only the first color from the color list will
be
 |              used.
 |
 |          - A dict of the form {column name : color}, so that each column will
be
 |              colored accordingly. For example, if your columns are called `a`
and
 |              `b`, then passing {'a': 'green', 'b': 'red'} will color bars for
 |              column `a` in green and bars for column `b` in red.
```

```
 |
 |          .. versionadded:: 1.1.0
 |
 |      **kwargs
 |          Additional keyword arguments are documented in
 |          :meth:`DataFrame.plot`.
 |
 |      Returns
 |      -------
 |      matplotlib.axes.Axes or np.ndarray of them
 |          An ndarray is returned with one :class:`matplotlib.axes.Axes`
 |          per column when ``subplots=True``.
 |
 |              See Also
 |              --------
 |              DataFrame.plot.bar: Vertical bar plot.
 |              DataFrame.plot : Make plots of DataFrame using matplotlib.
 |              matplotlib.axes.Axes.bar : Plot a vertical bar plot using
matplotlib.
 |
 |              Examples
 |              --------
 |              Basic example
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> df = pd.DataFrame({'lab': ['A', 'B', 'C'], 'val': [10,
30, 20]})
 |                  >>> ax = df.plot.barh(x='lab', y='val')
 |
 |              Plot a whole DataFrame to a horizontal bar plot
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
 |                  >>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
 |                  >>> index = ['snail', 'pig', 'elephant',
 |                  ...          'rabbit', 'giraffe', 'coyote', 'horse']
 |                  >>> df = pd.DataFrame({'speed': speed,
 |                  ...                     'lifespan': lifespan}, index=index)
 |                  >>> ax = df.plot.barh()
 |
 |              Plot stacked barh charts for the DataFrame
 |
 |              .. plot::
 |                  :context: close-figs
```

```
 |
 |                  >>> ax = df.plot.barh(stacked=True)
 |
 |              We can specify colors for each column
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> ax = df.plot.barh(color={"speed": "red", "lifespan":
"green"})
 |
 |              Plot a column of the DataFrame to a horizontal bar plot
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
 |                  >>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
 |                  >>> index = ['snail', 'pig', 'elephant',
 |                  ...           'rabbit', 'giraffe', 'coyote', 'horse']
 |                  >>> df = pd.DataFrame({'speed': speed,
 |                  ...                     'lifespan': lifespan}, index=index)
 |                  >>> ax = df.plot.barh(y='speed')
 |
 |              Plot DataFrame versus the desired column
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> speed = [0.1, 17.5, 40, 48, 52, 69, 88]
 |                  >>> lifespan = [2, 8, 70, 1.5, 25, 12, 28]
 |                  >>> index = ['snail', 'pig', 'elephant',
 |                  ...           'rabbit', 'giraffe', 'coyote', 'horse']
 |                  >>> df = pd.DataFrame({'speed': speed,
 |                  ...                     'lifespan': lifespan}, index=index)
 |                  >>> ax = df.plot.barh(x='lifespan')
 |
 |  box(self, by=None, **kwargs) -> 'PlotAccessor'
 |      Make a box plot of the DataFrame columns.
 |
 |      A box plot is a method for graphically depicting groups of numerical
 |      data through their quartiles.
 |      The box extends from the Q1 to Q3 quartile values of the data,
 |      with a line at the median (Q2). The whiskers extend from the edges
 |      of box to show the range of the data. The position of the whiskers
 |      is set by default to 1.5*IQR (IQR = Q3 - Q1) from the edges of the
 |      box. Outlier points are those past the end of the whiskers.
 |
```

```
|       For further details see Wikipedia's
|       entry for `boxplot <https://en.wikipedia.org/wiki/Box_plot>`__.
|
|       A consideration when using this chart is that the box and the whiskers
|       can overlap, which is very common when plotting small sets of data.
|
|       Parameters
|       ----------
|       by : str or sequence
|           Column in the DataFrame to group by.
|
|           .. versionchanged:: 1.4.0
|
|              Previously, `by` is silently ignore and makes no groupings
|
|       **kwargs
|           Additional keywords are documented in
|           :meth:`DataFrame.plot`.
|
|       Returns
|       -------
|       :class:`matplotlib.axes.Axes` or numpy.ndarray of them
|
|       See Also
|       --------
|       DataFrame.boxplot: Another method to draw a box plot.
|       Series.plot.box: Draw a box plot from a Series object.
|       matplotlib.pyplot.boxplot: Draw a box plot in matplotlib.
|
|       Examples
|       --------
|       Draw a box plot from a DataFrame with four columns of randomly
|       generated data.
|
|       .. plot::
|           :context: close-figs
|
|           >>> data = np.random.randn(25, 4)
|           >>> df = pd.DataFrame(data, columns=list('ABCD'))
|           >>> ax = df.plot.box()
|
|       You can also generate groupings if you specify the `by` parameter (which
|       can take a column name, or a list or tuple of column names):
|
|       .. versionchanged:: 1.4.0
|
|       .. plot::
|           :context: close-figs
```

```
 |
 |          >>> age_list = [8, 10, 12, 14, 72, 74, 76, 78, 20, 25, 30, 35, 60,
85]
 |          >>> df = pd.DataFrame({"gender": list("MMMMMMMMFFFFFF"), "age":
age_list})
 |          >>> ax = df.plot.box(column="age", by="gender", figsize=(10, 8))
 |
 |  density = kde(self, bw_method=None, ind=None, **kwargs) -> 'PlotAccessor'
 |
 |  hexbin(self, x, y, C=None, reduce_C_function=None, gridsize=None, **kwargs)
-> 'PlotAccessor'
 |      Generate a hexagonal binning plot.
 |
 |      Generate a hexagonal binning plot of `x` versus `y`. If `C` is `None`
 |      (the default), this is a histogram of the number of occurrences
 |      of the observations at ``(x[i], y[i])``.
 |
 |      If `C` is specified, specifies values at given coordinates
 |      ``(x[i], y[i])``. These values are accumulated for each hexagonal
 |      bin and then reduced according to `reduce_C_function`,
 |      having as default the NumPy's mean function (:meth:`numpy.mean`).
 |      (If `C` is specified, it must also be a 1-D sequence
 |      of the same length as `x` and `y`, or a column label.)
 |
 |      Parameters
 |      ----------
 |      x : int or str
 |          The column label or position for x points.
 |      y : int or str
 |          The column label or position for y points.
 |      C : int or str, optional
 |          The column label or position for the value of `(x, y)` point.
 |      reduce_C_function : callable, default `np.mean`
 |          Function of one argument that reduces all the values in a bin to
 |          a single number (e.g. `np.mean`, `np.max`, `np.sum`, `np.std`).
 |      gridsize : int or tuple of (int, int), default 100
 |          The number of hexagons in the x-direction.
 |          The corresponding number of hexagons in the y-direction is
 |          chosen in a way that the hexagons are approximately regular.
 |          Alternatively, gridsize can be a tuple with two elements
 |          specifying the number of hexagons in the x-direction and the
 |          y-direction.
 |      **kwargs
 |          Additional keyword arguments are documented in
 |          :meth:`DataFrame.plot`.
 |
 |      Returns
 |      -------
```

```
|      matplotlib.AxesSubplot
|          The matplotlib ``Axes`` on which the hexbin is plotted.
|
|      See Also
|      --------
|      DataFrame.plot : Make plots of a DataFrame.
|      matplotlib.pyplot.hexbin : Hexagonal binning plot using matplotlib,
|          the matplotlib function that is used under the hood.
|
|      Examples
|      --------
|      The following examples are generated with random data from
|      a normal distribution.
|
|      .. plot::
|          :context: close-figs
|
|          >>> n = 10000
|          >>> df = pd.DataFrame({'x': np.random.randn(n),
|          ...                    'y': np.random.randn(n)})
|          >>> ax = df.plot.hexbin(x='x', y='y', gridsize=20)
|
|      The next example uses `C` and `np.sum` as `reduce_C_function`.
|      Note that `'observations'` values ranges from 1 to 5 but the result
|      plot shows values up to more than 25. This is because of the
|      `reduce_C_function`.
|
|      .. plot::
|          :context: close-figs
|
|          >>> n = 500
|          >>> df = pd.DataFrame({
|          ...     'coord_x': np.random.uniform(-3, 3, size=n),
|          ...     'coord_y': np.random.uniform(30, 50, size=n),
|          ...     'observations': np.random.randint(1,5, size=n)
|          ...     })
|          >>> ax = df.plot.hexbin(x='coord_x',
|          ...                     y='coord_y',
|          ...                     C='observations',
|          ...                     reduce_C_function=np.sum,
|          ...                     gridsize=10,
|          ...                     cmap="viridis")
|
|  hist(self, by=None, bins: 'int' = 10, **kwargs) -> 'PlotAccessor'
|      Draw one histogram of the DataFrame's columns.
|
|      A histogram is a representation of the distribution of data.
|      This function groups the values of all given Series in the DataFrame
```

```
 |      into bins and draws all bins in one :class:`matplotlib.axes.Axes`.
 |      This is useful when the DataFrame's Series are in a similar scale.
 |
 |      Parameters
 |      ----------
 |      by : str or sequence, optional
 |          Column in the DataFrame to group by.
 |
 |          .. versionchanged:: 1.4.0
 |
 |             Previously, `by` is silently ignore and makes no groupings
 |
 |      bins : int, default 10
 |          Number of histogram bins to be used.
 |      **kwargs
 |          Additional keyword arguments are documented in
 |          :meth:`DataFrame.plot`.
 |
 |      Returns
 |      -------
 |      class:`matplotlib.AxesSubplot`
 |          Return a histogram plot.
 |
 |      See Also
 |      --------
 |      DataFrame.hist : Draw histograms per DataFrame's Series.
 |      Series.hist : Draw a histogram with Series' data.
 |
 |      Examples
 |      --------
 |      When we roll a die 6000 times, we expect to get each value around 1000
 |      times. But when we roll two dice and sum the result, the distribution
 |      is going to be quite different. A histogram illustrates those
 |      distributions.
 |
 |      .. plot::
 |          :context: close-figs
 |
 |          >>> df = pd.DataFrame(
 |          ...     np.random.randint(1, 7, 6000),
 |          ...     columns = ['one'])
 |          >>> df['two'] = df['one'] + np.random.randint(1, 7, 6000)
 |          >>> ax = df.plot.hist(bins=12, alpha=0.5)
 |
 |      A grouped histogram can be generated by providing the parameter `by`
(which
 |      can be a column name, or a list of column names):
 |
```

```
 |      .. plot::
 |          :context: close-figs
 |
 |          >>> age_list = [8, 10, 12, 14, 72, 74, 76, 78, 20, 25, 30, 35, 60,
85]
 |          >>> df = pd.DataFrame({"gender": list("MMMMMMMMFFFFFF"), "age":
age_list})
 |          >>> ax = df.plot.hist(column=["age"], by="gender", figsize=(10, 8))
 |
 |  kde(self, bw_method=None, ind=None, **kwargs) -> 'PlotAccessor'
 |      Generate Kernel Density Estimate plot using Gaussian kernels.
 |
 |      In statistics, `kernel density estimation`_ (KDE) is a non-parametric
 |      way to estimate the probability density function (PDF) of a random
 |      variable. This function uses Gaussian kernels and includes automatic
 |      bandwidth determination.
 |
 |      .. _kernel density estimation:
 |          https://en.wikipedia.org/wiki/Kernel_density_estimation
 |
 |      Parameters
 |      ----------
 |      bw_method : str, scalar or callable, optional
 |          The method used to calculate the estimator bandwidth. This can be
 |          'scott', 'silverman', a scalar constant or a callable.
 |          If None (default), 'scott' is used.
 |          See :class:`scipy.stats.gaussian_kde` for more information.
 |      ind : NumPy array or int, optional
 |          Evaluation points for the estimated PDF. If None (default),
 |          1000 equally spaced points are used. If `ind` is a NumPy array, the
 |          KDE is evaluated at the points passed. If `ind` is an integer,
 |          `ind` number of equally spaced points are used.
 |      **kwargs
 |          Additional keyword arguments are documented in
 |          :meth:`DataFrame.plot`.
 |
 |      Returns
 |      -------
 |      matplotlib.axes.Axes or numpy.ndarray of them
 |
 |      See Also
 |      --------
 |      scipy.stats.gaussian_kde : Representation of a kernel-density
 |          estimate using Gaussian kernels. This is the function used
 |          internally to estimate the PDF.
 |
 |      Examples
 |      --------
```

```
|       Given a Series of points randomly sampled from an unknown
|       distribution, estimate its PDF using KDE with automatic
|       bandwidth determination and plot the results, evaluating them at
|       1000 equally spaced points (default):
|
|       .. plot::
|           :context: close-figs
|
|           >>> s = pd.Series([1, 2, 2.5, 3, 3.5, 4, 5])
|           >>> ax = s.plot.kde()
|
|       A scalar bandwidth can be specified. Using a small bandwidth value can
|       lead to over-fitting, while using a large bandwidth value may result
|       in under-fitting:
|
|       .. plot::
|           :context: close-figs
|
|           >>> ax = s.plot.kde(bw_method=0.3)
|
|       .. plot::
|           :context: close-figs
|
|           >>> ax = s.plot.kde(bw_method=3)
|
|       Finally, the `ind` parameter determines the evaluation points for the
|       plot of the estimated PDF:
|
|       .. plot::
|           :context: close-figs
|
|           >>> ax = s.plot.kde(ind=[1, 2, 3, 4, 5])
|
|       For DataFrame, it works in the same way:
|
|       .. plot::
|           :context: close-figs
|
|           >>> df = pd.DataFrame({
|           ...     'x': [1, 2, 2.5, 3, 3.5, 4, 5],
|           ...     'y': [4, 4, 4.5, 5, 5.5, 6, 6],
|           ... })
|           >>> ax = df.plot.kde()
|
|       A scalar bandwidth can be specified. Using a small bandwidth value can
|       lead to over-fitting, while using a large bandwidth value may result
|       in under-fitting:
|
```

```
 |      .. plot::
 |          :context: close-figs
 |
 |          >>> ax = df.plot.kde(bw_method=0.3)
 |
 |      .. plot::
 |          :context: close-figs
 |
 |          >>> ax = df.plot.kde(bw_method=3)
 |
 |      Finally, the `ind` parameter determines the evaluation points for the
 |      plot of the estimated PDF:
 |
 |      .. plot::
 |          :context: close-figs
 |
 |          >>> ax = df.plot.kde(ind=[1, 2, 3, 4, 5, 6])
 |
 |  line(self, x=None, y=None, **kwargs) -> 'PlotAccessor'
 |      Plot Series or DataFrame as lines.
 |
 |      This function is useful to plot lines using DataFrame's values
 |      as coordinates.
 |
 |      Parameters
 |      ----------
 |      x : label or position, optional
 |          Allows plotting of one column versus another. If not specified,
 |          the index of the DataFrame is used.
 |      y : label or position, optional
 |          Allows plotting of one column versus another. If not specified,
 |          all numerical columns are used.
 |      color : str, array-like, or dict, optional
 |          The color for each of the DataFrame's columns. Possible values are:
 |
 |          - A single color string referred to by name, RGB or RGBA code,
 |              for instance 'red' or '#a98d19'.
 |
 |          - A sequence of color strings referred to by name, RGB or RGBA
 |              code, which will be used for each column recursively. For
 |              instance ['green','yellow'] each column's line will be filled in
 |              green or yellow, alternatively. If there is only a single column
to
 |              be plotted, then only the first color from the color list will
be
 |              used.
 |
 |          - A dict of the form {column name : color}, so that each column will
```

```
be
 |              colored accordingly. For example, if your columns are called `a`
and
 |              `b`, then passing {'a': 'green', 'b': 'red'} will color lines
for
 |              column `a` in green and lines for column `b` in red.
 |
 |          .. versionadded:: 1.1.0
 |
 |      **kwargs
 |          Additional keyword arguments are documented in
 |          :meth:`DataFrame.plot`.
 |
 |      Returns
 |      -------
 |      matplotlib.axes.Axes or np.ndarray of them
 |          An ndarray is returned with one :class:`matplotlib.axes.Axes`
 |          per column when ``subplots=True``.
 |
 |              See Also
 |              --------
 |              matplotlib.pyplot.plot : Plot y versus x as lines and/or
markers.
 |
 |              Examples
 |              --------
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  >>> s = pd.Series([1, 3, 2])
 |                  >>> s.plot.line()
 |                  <AxesSubplot: ylabel='Density'>
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  The following example shows the populations for some animals
 |                  over the years.
 |
 |                  >>> df = pd.DataFrame({
 |                  ...    'pig': [20, 18, 489, 675, 1776],
 |                  ...    'horse': [4, 25, 281, 600, 1900]
 |                  ...    }, index=[1990, 1997, 2003, 2009, 2014])
 |                  >>> lines = df.plot.line()
 |
 |              .. plot::
 |                 :context: close-figs
```

```
 |
 |                  An example with subplots, so an array of axes is returned.
 |
 |                  >>> axes = df.plot.line(subplots=True)
 |                  >>> type(axes)
 |                  <class 'numpy.ndarray'>
 |
 |              .. plot::
 |                  :context: close-figs
 |
 |                  Let's repeat the same example, but specifying colors for
 |                  each column (in this case, for each animal).
 |
 |                  >>> axes = df.plot.line(
 |                  ...     subplots=True, color={"pig": "pink", "horse":
"#742802"}
 |                  ... )
 |
 |              .. plot::
 |                   :context: close-figs
 |
 |                   The following example shows the relationship between both
 |                   populations.
 |
 |                   >>> lines = df.plot.line(x='pig', y='horse')
 |
 |  pie(self, **kwargs) -> 'PlotAccessor'
 |      Generate a pie plot.
 |
 |      A pie plot is a proportional representation of the numerical data in a
 |      column. This function wraps :meth:`matplotlib.pyplot.pie` for the
 |      specified column. If no column reference is passed and
 |      ``subplots=True`` a pie plot is drawn for each numerical column
 |      independently.
 |
 |      Parameters
 |      ----------
 |      y : int or label, optional
 |          Label or position of the column to plot.
 |          If not provided, ``subplots=True`` argument must be passed.
 |      **kwargs
 |          Keyword arguments to pass on to :meth:`DataFrame.plot`.
 |
 |      Returns
 |      -------
 |      matplotlib.axes.Axes or np.ndarray of them
 |          A NumPy array is returned when `subplots` is True.
 |
```

```
|      See Also
|      --------
|      Series.plot.pie : Generate a pie plot for a Series.
|      DataFrame.plot : Make plots of a DataFrame.
|
|      Examples
|      --------
|      In the example below we have a DataFrame with the information about
|      planet's mass and radius. We pass the 'mass' column to the
|      pie function to get a pie plot.
|
|      .. plot::
|          :context: close-figs
|
|          >>> df = pd.DataFrame({'mass': [0.330, 4.87 , 5.97],
|          ...                    'radius': [2439.7, 6051.8, 6378.1]},
|          ...                   index=['Mercury', 'Venus', 'Earth'])
|          >>> plot = df.plot.pie(y='mass', figsize=(5, 5))
|
|      .. plot::
|          :context: close-figs
|
|          >>> plot = df.plot.pie(subplots=True, figsize=(11, 6))
|
|  scatter(self, x, y, s=None, c=None, **kwargs) -> 'PlotAccessor'
|      Create a scatter plot with varying marker point size and color.
|
|      The coordinates of each point are defined by two dataframe columns and
|      filled circles are used to represent each point. This kind of plot is
|      useful to see complex correlations between two variables. Points could
|      be for instance natural 2D coordinates like longitude and latitude in
|      a map or, in general, any pair of metrics that can be plotted against
|      each other.
|
|      Parameters
|      ----------
|      x : int or str
|          The column name or column position to be used as horizontal
|          coordinates for each point.
|      y : int or str
|          The column name or column position to be used as vertical
|          coordinates for each point.
|      s : str, scalar or array-like, optional
|          The size of each point. Possible values are:
|
|          - A string with the name of the column to be used for marker's size.
|
|          - A single scalar so all points have the same size.
```

```
|
|          - A sequence of scalars, which will be used for each point's size
|            recursively. For instance, when passing [2,14] all points size
|            will be either 2 or 14, alternatively.
|
|            .. versionchanged:: 1.1.0
|
|      c : str, int or array-like, optional
|          The color of each point. Possible values are:
|
|          - A single color string referred to by name, RGB or RGBA code,
|            for instance 'red' or '#a98d19'.
|
|          - A sequence of color strings referred to by name, RGB or RGBA
|            code, which will be used for each point's color recursively. For
|            instance ['green','yellow'] all points will be filled in green or
|            yellow, alternatively.
|
|          - A column name or position whose values will be used to color the
|            marker points according to a colormap.
|
|      **kwargs
|          Keyword arguments to pass on to :meth:`DataFrame.plot`.
|
|      Returns
|      -------
|      :class:`matplotlib.axes.Axes` or numpy.ndarray of them
|
|      See Also
|      --------
|      matplotlib.pyplot.scatter : Scatter plot using multiple input data
|          formats.
|
|      Examples
|      --------
|      Let's see how to draw a scatter plot using coordinates from the values
|      in a DataFrame's columns.
|
|      .. plot::
|          :context: close-figs
|
|          >>> df = pd.DataFrame([[5.1, 3.5, 0], [4.9, 3.0, 0], [7.0, 3.2, 1],
|          ...                    [6.4, 3.2, 1], [5.9, 3.0, 2]],
|          ...                   columns=['length', 'width', 'species'])
|          >>> ax1 = df.plot.scatter(x='length',
|          ...                       y='width',
|          ...                       c='DarkBlue')
|
```

```
|      And now with the color determined by a column as well.
|
|      .. plot::
|          :context: close-figs
|
|          >>> ax2 = df.plot.scatter(x='length',
|          ...                       y='width',
|          ...                       c='species',
|          ...                       colormap='viridis')
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
|
|  __annotations__ = {}
|
|  ----------------------------------------------------------------------
|  Methods inherited from pandas.core.base.PandasObject:
|
|  __repr__(self) -> 'str'
|      Return a string representation for a particular object.
|
|  __sizeof__(self) -> 'int'
|      Generates the total memory usage for an object that returns
|      either a value or Series of values
|
|  ----------------------------------------------------------------------
|  Methods inherited from pandas.core.accessor.DirNamesMixin:
|
|  __dir__(self) -> 'list[str]'
|      Provide method name lookup and completion.
|
|      Notes
|      -----
|      Only provide 'public' methods.
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from pandas.core.accessor.DirNamesMixin:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
```