

```
: df = pd.DataFrame({  
    "Name": ["Braund, Mr. Owen Harris", "Allen, Mr. William Henry", "Bonnell, Miss. Elizabeth"],  
    "Age": [22, 35, 58],  
    "Sex": ["male", "male", "female"]  
})  
df
```

	Name	Age	Sex
0	Braund, Mr. Owen Harris	22	male
1	Allen, Mr. William Henry	35	male
2	Bonnell, Miss. Elizabeth	58	female

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

```
In [2]: df = pd.DataFrame({  
  
    "Name": ["Braund, Mr. Owen Harris", "Allen, Mr. William Henry", "Bonnell, Miss.  
    Brown, Mrs. Johnson"],  
  
    "Age": [22, 35, 58],  
  
    "Sex": ["male", "male", "female"]  
  
})  
  
df
```

```
Out[2]:
```

	Name	Age	Sex
0	Braund, Mr. Owen Harris	22	male
1	Allen, Mr. William Henry	35	male
2	Bonnell, Miss. Elizabeth	58	female

```
In [3]: df["Age"]
```

```
Out[3]: 0    22  
1    35  
2    58  
Name: Age, dtype: int64
```

```
In [4]: ages = pd.Series([22, 35, 58], name="Age")  
ages
```

```
Out[4]: 0    22  
1    35  
2    58  
Name: Age, dtype: int64
```

```
In [5]: df["Age"].max()
```

```
Out[5]: 58
```

```
In [6]: ages.max()
```

```
Out[6]: 58
```

```
In [7]: df.describe()
```

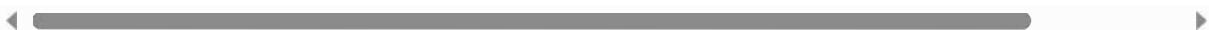
Out[7]:

	Age
count	3.000000
mean	38.333333
std	18.230012
min	22.000000
25%	28.500000
50%	35.000000
75%	46.500000
max	58.000000

```
In [8]: titanic = pd.read_csv("train_titanic.csv")
titanic.head()
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cal
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N



```
In [9]: titanic.dtypes
```

```
Out[9]: PassengerId      int64
Survived        int64
Pclass          int64
Name            object
Sex             object
Age             float64
SibSp           int64
Parch           int64
Ticket          object
Fare            float64
Cabin           object
Embarked        object
dtype: object
```

```
In [10]: titanic.to_excel("titanic.xlsx", sheet_name="passengers", index=False)
```

```
In [11]: titanic = pd.read_excel("titanic.xlsx", sheet_name="passengers")
```

```
In [12]: titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object  
 4   Sex          891 non-null    object  
 5   Age          714 non-null    float64 
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object  
 9   Fare         891 non-null    float64 
 10  Cabin        204 non-null    object  
 11  Embarked     889 non-null    object  
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
In [13]: ages = titanic["Age"]
ages.head()
```

```
Out[13]: 0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
Name: Age, dtype: float64
```

```
In [14]: type(titanic["Age"])
```

```
Out[14]: pandas.core.series.Series
```

```
In [15]: titanic["Age"].shape
```

```
Out[15]: (891,)
```

```
In [16]: titanic["Age"].shape
```

```
Out[16]: (891,)
```

```
In [17]: age_sex = titanic[["Age", "Sex"]]
age_sex.head()
```

```
Out[17]:
```

	Age	Sex
0	22.0	male
1	38.0	female
2	26.0	female
3	35.0	female
4	35.0	male

```
In [18]: titanic[["Age", "Sex"]].shape
```

```
Out[18]: (891, 2)
```

```
In [19]: above_35 = titanic[titanic["Age"] > 35]
above_35.head()
```

```
Out[19]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
1	1	2	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C
6	6	7	0	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E
11	11	12	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C10
13	13	14	0	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	N
15	15	16	1	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	N



```
In [20]: class_23 = titanic[titanic["Pclass"].isin([2, 3])]  
class_23.head()
```

Out[20]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cal
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	N
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	N

```
In [21]: class_23 = titanic[(titanic["Pclass"] == 2) | (titanic["Pclass"] == 3)]  
class_23.head()
```

Out[21]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cal
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	N
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	N

```
In [22]: age_no_na = titanic[titanic["Age"].notna()]
age_no_na.head()
```

```
Out[22]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	N
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	N
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C1
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	N

```
In [23]: adult_names = titanic.loc[titanic["Age"] > 35]
adult_names.head()
```

```
Out[23]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cab
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...)	female	38.0	1	0	PC 17599	71.2833	C
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E
11	12	1	1	Bonnell, Miss. Elizabeth	female	58.0	0	0	113783	26.5500	C10
13	14	0	3	Andersson, Mr. Anders Johan	male	39.0	1	5	347082	31.2750	N&
15	16	1	2	Hewlett, Mrs. (Mary D Kingcome)	female	55.0	0	0	248706	16.0000	N&

```
In [24]: adult_names = titanic.loc[titanic["Age"] > 35, "Name"]
adult_names.head()
```

```
Out[24]: 1    Cumings, Mrs. John Bradley (Florence Briggs Th...
6                McCarthy, Mr. Timothy J
11               Bonnell, Miss. Elizabeth
13          Andersson, Mr. Anders Johan
15      Hewlett, Mrs. (Mary D Kingcome)
Name: Name, dtype: object
```

```
In [25]: titanic.iloc[9:25, 2:5]
```

```
Out[25]:   Pclass          Name     Sex
  9      2    Nasser, Mrs. Nicholas (Adele Achem)  female
 10     3    Sandstrom, Miss. Marguerite Rut  female
 11     1    Bonnell, Miss. Elizabeth  female
 12     3  Saundercock, Mr. William Henry    male
 13     3    Andersson, Mr. Anders Johan    male
 14     3  Vestrom, Miss. Hulda Amanda Adolfina  female
 15     2    Hewlett, Mrs. (Mary D Kingcome)  female
 16     3        Rice, Master. Eugene    male
 17     2    Williams, Mr. Charles Eugene    male
 18     3  Vander Planke, Mrs. Julius (Emelia Maria Vande...)  female
 19     3    Masselmani, Mrs. Fatima  female
 20     2        Fynney, Mr. Joseph J    male
 21     2    Beesley, Mr. Lawrence    male
 22     3    McGowan, Miss. Anna "Annie"  female
 23     1    Sloper, Mr. William Thompson    male
 24     3    Palsson, Miss. Torborg Danira  female
```

```
In [26]: anon = titanic.iloc[0:3, 3] = "anonymous"
anon
```

```
Out[26]: 'anonymous'
```

```
In [27]: titanic.head()
```

```
Out[27]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	1	0	3	anonymous	male	22.0	1	0	A/5 21171	7.2500	I
1	2	1	1	anonymous	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	anonymous	female	26.0	0	0	STON/O2. 3101282	7.9250	I
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	I

```
In [28]: titanic["Age"].mean()
```

```
Out[28]: 29.69911764705882
```

```
In [29]: titanic[["Age", "Fare"]].median()
```

```
Out[29]: Age      28.0000
Fare     14.4542
dtype: float64
```

```
In [30]: titanic[["Age", "Fare"]].describe()
```

```
Out[30]:
```

	Age	Fare
count	714.000000	891.000000
mean	29.699118	32.204208
std	14.526497	49.693429
min	0.420000	0.000000
25%	20.125000	7.910400
50%	28.000000	14.454200
75%	38.000000	31.000000
max	80.000000	512.329200

```
In [31]: titanic.agg({  
    "Age": ["min", "max", "median", "skew"],  
    "Fare": ["min", "max", "median", "mean"]  
})
```

```
Out[31]:
```

	Age	Fare
min	0.420000	0.000000
max	80.000000	512.329200
median	28.000000	14.454200
skew	0.389108	Nan
mean	Nan	32.204208

```
In [32]: titanic[["Sex", "Age"]].groupby("Sex").mean()
```

```
Out[32]:
```

	Age
Sex	
female	27.915709
male	30.726645

```
In [33]: titanic[["Sex", "Age"]].groupby("Sex").max()
```

```
Out[33]:
```

	Age
Sex	
female	63.0
male	80.0

```
In [34]: titanic[["Sex", "Age"]].groupby("Sex").first()
```

```
Out[34]:
```

	Age
Sex	
female	38.0
male	22.0

```
In [35]: titanic.head(2)
```

```
Out[35]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	anonymous	male	22.0	1	0	A/5 21171	7.2500	Nan
1	2	1	1	anonymous	female	38.0	1	0	PC 17599	71.2833	C85

```
In [37]: titanic.groupby("Sex")["Age"].mean()
```

```
Out[37]: Sex
female    27.915709
male      30.726645
Name: Age, dtype: float64
```

```
In [38]: titanic.groupby(["Sex", "Pclass"])["Fare"].mean()
```

```
Out[38]: Sex      Pclass
female   1        106.125798
          2        21.970121
          3        16.118810
male     1        67.226127
          2        19.741782
          3        12.661633
Name: Fare, dtype: float64
```

```
In [39]: titanic["Pclass"].value_counts()
```

```
Out[39]: 3    491
1    216
2    184
Name: Pclass, dtype: int64
```

```
In [40]: titanic.groupby("Pclass")["Pclass"].count()
```

```
Out[40]: Pclass
1    216
2    184
3    491
Name: Pclass, dtype: int64
```

```
In [41]: titanic.sort_values(by="Age", ascending=False).head()
```

```
Out[41]:   PassengerId  Survived  Pclass           Name  Sex  Age  SibSp  Parch  Ticket  Fare  Ca
              630        631       1      1  Barkworth,  
                           Mr. Algernon  
                           Henry  
                           Wilson  male  80.0      0      0   27042  30.0000  A
              851        852       0      3  Svensson,  
                           Mr. Johan  male  74.0      0      0   347060  7.7750  N
              493        494       0      1  Artagaveytia,  
                           Mr. Ramon  male  71.0      0      0      PC  
                           17609  49.5042  N
              96         97       0      1  Goldschmidt,  
                           Mr. George  
                           B  male  71.0      0      0      PC  
                           17754  34.6542
             116        117       0      3  Connors, Mr.  
                           Patrick  male  70.5      0      0   370369  7.7500  N
```



```
In [42]: titanic.sort_values(by=['Pclass', 'Age'], ascending=False).head()
```

```
Out[42]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
851	852	0	3	Svensson, Mr. Johan	male	74.0	0	0	347060	7.7750	Nal
116	117	0	3	Connors, Mr. Patrick	male	70.5	0	0	370369	7.7500	Nal
280	281	0	3	Duane, Mr. Frank	male	65.0	0	0	336439	7.7500	Nal
483	484	1	3	Turkula, Mrs. (Hedwig)	female	63.0	0	0	4134	9.5875	Nal
326	327	0	3	Nysveen, Mr. Johan Hansen	male	61.0	0	0	345364	6.2375	Nal

```
In [43]: titanic.dtypes
```

```
Out[43]:
```

PassengerId	int64
Survived	int64
Pclass	int64
Name	object
Sex	object
Age	float64
SibSp	int64
Parch	int64
Ticket	object
Fare	float64
Cabin	object
Embarked	object
dtype:	object

```
In [44]: titanic["Name"].str.lower()
```

```
Out[44]:
```

0	anonymous
1	anonymous
2	anonymous
3	futrelle, mrs. jacques heath (lily may peel)
4	allen, mr. william henry
	...
886	montvila, rev. juozas
887	graham, miss. margaret edith
888	johnston, miss. catherine helen "carrie"
889	behr, mr. karl howell
890	dooley, mr. patrick
Name: Name, Length: 891, dtype: object	

```
In [45]: titanic["Name"].str.split(",")
```

```
Out[45]: 0                               [anonymous]
          1                               [anonymous]
          2                               [anonymous]
          3      [Futrelle, Mrs. Jacques Heath (Lily May Peel)]
          4          [Allen, Mr. William Henry]
          ...
          886          [Montvila, Rev. Juozas]
          887          [Graham, Miss. Margaret Edith]
          888          [Johnston, Miss. Catherine Helen "Carrie"]
          889          [Behr, Mr. Karl Howell]
          890          [Dooley, Mr. Patrick]
Name: Name, Length: 891, dtype: object
```

```
In [46]: titanic["Surname"] = titanic["Name"].str.split(",").str.get(0)
titanic["Surname"]
```

```
Out[46]: 0      anonymous
          1      anonymous
          2      anonymous
          3      Futrelle
          4      Allen
          ...
          886     Montvila
          887     Graham
          888     Johnston
          889     Behr
          890     Dooley
Name: Surname, Length: 891, dtype: object
```

```
In [47]: titanic["Name_main"] = titanic["Name"].str.split(",").str.get(1)
titanic["Name_main"]
```

```
Out[47]: 0                  NaN
          1                  NaN
          2                  NaN
          3      Mrs. Jacques Heath (Lily May Peel)
          4          Mr. William Henry
          ...
          886          Rev. Juozas
          887          Miss. Margaret Edith
          888          Miss. Catherine Helen "Carrie"
          889          Mr. Karl Howell
          890          Mr. Patrick
Name: Name_main, Length: 891, dtype: object
```

```
In [48]: titanic["Name"].str.split(",")
```

```
Out[48]: 0                               [anonymous]  
1                               [anonymous]  
2                               [anonymous]  
3      [Futrelle, Mrs. Jacques Heath (Lily May Peel)]  
4      [Allen, Mr. William Henry]  
...  
886      [Montvila, Rev. Juozas]  
887      [Graham, Miss. Margaret Edith]  
888      [Johnston, Miss. Catherine Helen "Carrie"]  
889      [Behr, Mr. Karl Howell]  
890      [Dooley, Mr. Patrick]  
Name: Name, Length: 891, dtype: object
```

```
In [49]: titanic['Real_Name'] = titanic["Name"].str.split(",").str.get(0)  
titanic.head()
```

```
Out[49]:   PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  Parch     Ticket    Fare  C  
0            1         0      3  anonymous   male  22.0      1      0  A/5 21171  7.2500  I  
1            2         1      1  anonymous  female  38.0      1      0   PC 17599  71.2833  
2            3         1      3  anonymous  female  26.0      0      0  STON/O2.  
3101282    7.9250  I  
3            4         1      1  Futrelle,  
                         Mrs.  
                         Jacques  female  35.0      1      0  113803  53.1000  C  
Heath (Lily  
May Peel)  
4            5         0      3  Allen, Mr.  
                         William  male  35.0      0      0  373450  8.0500  I  
                         Henry
```

```
In [50]: titanic['Surname'] = titanic["Name"].str.split(",").str.get(1)  
titanic.head()
```

```
Out[50]:   PassengerId  Survived  Pclass          Name     Sex   Age  SibSp  Parch     Ticket    Fare  C  
0            1         0      3  anonymous   male  22.0      1      0  A/5 21171  7.2500  I  
1            2         1      1  anonymous  female  38.0      1      0   PC 17599  71.2833  
2            3         1      3  anonymous  female  26.0      0      0  STON/O2.  
3101282    7.9250  I  
3            4         1      1  Futrelle,  
                         Mrs.  
                         Jacques  female  35.0      1      0  113803  53.1000  C  
Heath (Lily  
May Peel)  
4            5         0      3  Allen, Mr.  
                         William  male  35.0      0      0  373450  8.0500  I  
                         Henry
```

```
In [51]: titanic['Salutation'] = titanic['Surname'].str.split('.').str.get(0)
titanic.head()
```

```
Out[51]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
0	1	0	3	anonymous	male	22.0	1	0	A/5 21171	7.2500	I
1	2	1	1	anonymous	female	38.0	1	0	PC 17599	71.2833	
2	3	1	3	anonymous	female	26.0	0	0	STON/O2. 3101282	7.9250	I
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	I

```
In [52]: titanic["Name"].str.contains("Mr")
```

```
Out[52]: 0      False
1      False
2      False
3      True
4      True
...
886    False
887    False
888    False
889    True
890    True
Name: Name, Length: 891, dtype: bool
```

```
In [53]: titanic[titanic["Name"].str.contains("Countess")]
```

```
Out[53]:
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
759	760	1	1	Rothes, the Countess. of (Lucy Noel Martha Dye...)	female	33.0	0	0	110152	86.5	B77

```
In [54]: titanic["Name"].str.len()
```

```
Out[54]: 0      9  
1      9  
2      9  
3     44  
4     24  
..  
886    21  
887    28  
888    40  
889    21  
890    19  
Name: Name, Length: 891, dtype: int64
```

```
In [55]: titanic["Name"].str.len().idxmax()
```

```
Out[55]: 307
```

```
In [56]: titanic.loc[titanic["Name"].str.len().idxmax(), "Name"]
```

```
Out[56]: 'Penasco y Castellana, Mrs. Victor de Satode (Maria Josefa Perez de Soto y Vallejo)'
```

```
In [57]: titanic.loc[titanic["Name"].str.len().idxmin(), "Name"]
```

```
Out[57]: 'anonymous'
```

```
In [58]: titanic["Sex_short"] = titanic["Sex"].replace({"male": "M", "female": "F"})  
titanic["Sex_short"]
```

```
Out[58]: 0      M  
1      F  
2      F  
3      F  
4      M  
..  
886    M  
887    F  
888    F  
889    M  
890    M  
Name: Sex_short, Length: 891, dtype: object
```

```
In [59]: titanic["Sex_short"] = titanic["Sex"].str.replace("female", "F")  
titanic["Sex_short"] = titanic["Sex_short"].str.replace("male", "M")
```

```
In [170]: import numpy as np  
df = pd.DataFrame(np.random.randn(10, 3), columns=list("abc"))  
df[["a", "c", "b"]]
```

Out[170]:

	a	c	b
0	0.971377	-0.762178	-0.305884
1	0.412251	0.588495	0.096369
2	1.801618	-0.597973	1.489147
3	-0.359858	-0.878680	-1.461579
4	-0.455795	0.681250	0.973445
5	0.852787	-0.544525	-0.295961
6	-1.098355	-1.421945	-0.417816
7	-0.133820	-0.183852	1.228257
8	-0.495825	-1.226723	-0.318924
9	-0.064218	-0.306832	-0.345931

```
In [171]: df.loc[:, ["a", "c"]]
```

Out[171]:

	a	c
0	0.971377	-0.762178
1	0.412251	0.588495
2	1.801618	-0.597973
3	-0.359858	-0.878680
4	-0.455795	0.681250
5	0.852787	-0.544525
6	-1.098355	-1.421945
7	-0.133820	-0.183852
8	-0.495825	-1.226723
9	-0.064218	-0.306832

Good Code

```
In [ ]: named = list("abcdefg")  
n = 30  
columns = named + np.arange(len(named), n).tolist()  
df = pd.DataFrame(np.random.randn(n, n), columns=columns)  
df.iloc[:, np.r_[:10, 24:30]]
```

```
In [ ]: df = pd.DataFrame({  
  
    "v1": [1, 3, 5, 7, 8, 3, 5, np.nan, 4, 5, 7, 9],  
    "v2": [11, 33, 55, 77, 88, 33, 55, np.nan, 44, 55, 77, 99],  
    "by1": ["red", "blue", 1, 2, np.nan, "big", 1, 2, "red", 1, np.nan, 12],  
    "by2": ["wet", "dry", 99, 95, np.nan, "damp", 95, 99, "red", 99, np.nan, np.nan,]  
  
})  
  
df
```

```
In [ ]: g = df.groupby(["by1", "by2"])  
g[["v1", "v2"]].mean()
```

```
In [63]: import numpy as np  
s = pd.Series(np.arange(5), dtype=np.float32)  
s
```

```
Out[63]: 0    0.0  
1    1.0  
2    2.0  
3    3.0  
4    4.0  
dtype: float32
```

```
In [64]: s.isin([2, 4])
```

```
Out[64]: 0    False  
1    False  
2    True  
3    False  
4    True  
dtype: bool
```

Data generation code

```
In [65]: # Data generation code
```

```
import random
import string

baseball = pd.DataFrame({
    "team": ["team %d" % (x + 1) for x in range(5)] * 5,
    "player": random.sample(list(string.ascii_lowercase), 25),
    "batting avg": np.random.uniform(0.200, 0.400, 25),
})
baseball
```

```
Out[65]:
```

	team	player	batting avg
0	team 1	b	0.311944
1	team 2	w	0.300678
2	team 3	c	0.271453
3	team 4	p	0.301531
4	team 5	a	0.257927
5	team 1	q	0.384259
6	team 2	d	0.279827
7	team 3	s	0.200344
8	team 4	f	0.269042
9	team 5	k	0.363716
10	team 1	u	0.355087
11	team 2	v	0.276580
12	team 3	z	0.381452
13	team 4	g	0.264230
14	team 5	e	0.397186
15	team 1	n	0.249416
16	team 2	i	0.245684
17	team 3	y	0.316917
18	team 4	o	0.206810
19	team 5	m	0.272293
20	team 1	h	0.328023
21	team 2	r	0.352936
22	team 3	t	0.350134
23	team 4	x	0.368002
24	team 5	j	0.282423

```
In [66]: baseball.pivot_table(values="batting avg", columns="team", aggfunc=np.max)
```

```
Out[66]:
```

	team	team 1	team 2	team 3	team 4	team 5
batting avg	0.384259	0.352936	0.381452	0.368002	0.397186	

```
In [67]: df = pd.DataFrame({"a": np.random.randn(10), "b": np.random.randn(10)})  
df.head()
```

```
Out[67]:
```

	a	b
0	0.848214	1.528596
1	-1.363807	-1.321466
2	-0.525568	1.252385
3	-0.351879	-0.315065
4	-0.700257	0.328759

```
In [68]: df.query("a <= b")
```

```
Out[68]:
```

	a	b
0	0.848214	1.528596
1	-1.363807	-1.321466
2	-0.525568	1.252385
3	-0.351879	-0.315065
4	-0.700257	0.328759
5	-0.927895	-0.516451
6	-1.526683	-0.259954
7	-2.127308	0.531293
8	-0.206562	0.237485
9	0.413750	0.770686

```
In [69]: df[df["a"] <= df["b"]]
```

Out[69]:

	a	b
0	0.848214	1.528596
1	-1.363807	-1.321466
2	-0.525568	1.252385
3	-0.351879	-0.315065
4	-0.700257	0.328759
5	-0.927895	-0.516451
6	-1.526683	-0.259954
7	-2.127308	0.531293
8	-0.206562	0.237485
9	0.413750	0.770686

```
In [70]: df.loc[df["a"] <= df["b"]]
```

Out[70]:

	a	b
0	0.848214	1.528596
1	-1.363807	-1.321466
2	-0.525568	1.252385
3	-0.351879	-0.315065
4	-0.700257	0.328759
5	-0.927895	-0.516451
6	-1.526683	-0.259954
7	-2.127308	0.531293
8	-0.206562	0.237485
9	0.413750	0.770686

```
In [71]: df[df["a"] >= df["b"]]
```

Out[71]:

	a	b
--	---	---

```
In [72]: df = pd.DataFrame({"a": np.random.randn(10), "b": np.random.randn(10)})  
df.head()
```

```
Out[72]:
```

	a	b
0	0.057788	-0.548498
1	-0.150495	-1.303927
2	0.391174	-0.383887
3	-0.486376	0.660384
4	-0.149571	0.048288

```
In [73]: df.eval("a + b")
```

```
Out[73]: 0    -0.490711  
1    -1.454421  
2     0.007287  
3     0.174008  
4    -0.101283  
5    -0.636180  
6     0.540169  
7    -0.429076  
8     0.185766  
9    -1.048530  
dtype: float64
```

```
In [74]: df["a"] + df["b"]
```

```
Out[74]: 0    -0.490711  
1    -1.454421  
2     0.007287  
3     0.174008  
4    -0.101283  
5    -0.636180  
6     0.540169  
7    -0.429076  
8     0.185766  
9    -1.048530  
dtype: float64
```

```
In [75]: df = pd.DataFrame({
    "x": np.random.uniform(1.0, 168.0, 120),
    "y": np.random.uniform(7.0, 334.0, 120),
    "z": np.random.uniform(1.7, 20.7, 120),
    "month": [5, 6, 7, 8] * 30,
    "week": np.random.randint(1, 4, 120)
})
df.head()
```

Out[75]:

	x	y	z	month	week
0	41.516759	33.530167	18.106587	5	1
1	35.505487	252.682232	14.136898	6	3
2	91.041404	170.303748	7.498431	7	2
3	26.488195	332.594130	5.038641	8	3
4	105.767124	107.686286	15.308504	5	2

In [76]: grouped = df.groupby(["month", "week"])
grouped["x"].agg([np.mean, np.std])

Out[76]:

	mean	std
month week		
1 1	87.949769	52.340418
5 2	80.472147	51.547185
5 3	83.919926	38.707154
1 1	62.640842	45.863364
6 2	76.669235	49.722219
6 3	74.546488	45.781976
1 1	83.765432	35.884936
7 2	120.548997	36.888050
7 3	90.085512	49.915038
1 1	85.325932	60.874941
8 2	50.293565	37.083210
8 3	59.143017	41.149256

```
In [77]: a = np.array(list(range(1, 24)) + [np.NAN]).reshape(2, 3, 4)
a
```

```
Out[77]: array([[[ 1.,  2.,  3.,  4.],
   [ 5.,  6.,  7.,  8.],
   [ 9., 10., 11., 12.]],

  [[13., 14., 15., 16.],
   [17., 18., 19., 20.],
   [21., 22., 23., nan]]])
```

```
In [78]: pd.DataFrame([tuple(list(x) + [val]) for x, val in np.ndenumerate(a)])
```

```
Out[78]:   0  1  2    3
0  0  0  0  1.0
1  0  0  1  2.0
2  0  0  2  3.0
3  0  0  3  4.0
4  0  1  0  5.0
5  0  1  1  6.0
6  0  1  2  7.0
7  0  1  3  8.0
8  0  2  0  9.0
9  0  2  1  10.0
10  0  2  2  11.0
11  0  2  3  12.0
12  1  0  0  13.0
13  1  0  1  14.0
14  1  0  2  15.0
15  1  0  3  16.0
16  1  1  0  17.0
17  1  1  1  18.0
18  1  1  2  19.0
19  1  1  3  20.0
20  1  2  0  21.0
21  1  2  1  22.0
22  1  2  2  23.0
23  1  2  3  NaN
```

```
In [79]: a = list(enumerate(list(range(1, 5)) + [np.NAN]))  
a
```

```
Out[79]: [(0, 1), (1, 2), (2, 3), (3, 4), (4, nan)]
```

```
In [80]: pd.DataFrame(a)
```

```
Out[80]:   0    1  
0  0  1.0  
1  1  2.0  
2  2  3.0  
3  3  4.0  
4  4  NaN
```

```
In [81]: cheese = pd.DataFrame({  
  
    "first": ["John", "Mary"],  
    "last": ["Doe", "Bo"],  
    "height": [5.5, 6.0],  
    "weight": [130, 150]  
  
})  
  
cheese
```

```
Out[81]:   first  last  height  weight  
0  John   Doe      5.5     130  
1  Mary    Bo      6.0     150
```

```
In [82]: pd.melt(cheese, id_vars=["first", "last"])
```

```
Out[82]:   first  last  variable  value  
0  John   Doe    height    5.5  
1  Mary    Bo    height    6.0  
2  John   Doe   weight   130.0  
3  Mary    Bo   weight   150.0
```

```
In [83]: cheese.set_index(["first", "last"]).stack() # alternative
```

```
Out[83]: first  last  
John    Doe    height      5.5  
                  weight    130.0  
Mary    Bo    height      6.0  
                  weight    150.0  
dtype: float64
```

```
In [84]: df = pd.DataFrame({
    "x": np.random.uniform(1.0, 168.0, 12),
    "y": np.random.uniform(7.0, 334.0, 12),
    "z": np.random.uniform(1.7, 20.7, 12),
    "month": [5, 6, 7] * 4,
    "week": [1, 2] * 6
})

mdf = pd.melt(df, id_vars=["month", "week"])

pd.pivot_table(mdf, values="value", index=["variable", "week"], columns=["month"],
```

Out[84]:

	month	5	6	7
variable	week			
x	1	69.688604	58.926280	50.639441
	2	67.470350	117.676001	118.517232
y	1	160.009684	21.384183	177.609046
	2	203.316298	197.839213	184.583499
z	1	10.060922	4.185807	8.656566
	2	8.009302	11.105621	11.506984

```
In [85]: df = pd.DataFrame({
    "Animal": ["Animal1", "Animal2", "Animal3", "Animal2", "Animal1", "Animal2", "Animal3"],
    "FeedType": ["A", "B", "A", "A", "B", "B", "A"],
    "Amount": [10, 7, 4, 2, 5, 6, 2]
})

df.pivot_table(values="Amount", index="Animal", columns="FeedType", aggfunc="su
```

Out[85]:

FeedType	A	B
Animal		
Animal1	10.0	5.0
Animal2	2.0	13.0
Animal3	6.0	NaN

```
In [86]: df.groupby(["Animal", "FeedType"])["Amount"].sum()
```

Out[86]:

Animal	FeedType	Amount
Animal1	A	10
	B	5
Animal2	A	2
	B	13
Animal3	A	6

Name: Amount, dtype: int64

```
In [87]: pd.cut(pd.Series([1, 2, 3, 4, 5, 6]), 3)
```

```
Out[87]: 0    (0.995, 2.667]
         1    (0.995, 2.667]
         2    (2.667, 4.333]
         3    (2.667, 4.333]
         4    (4.333, 6.0]
         5    (4.333, 6.0]
        dtype: category
Categories (3, interval[float64, right]): [(0.995, 2.667] < (2.667, 4.333] <
(4.333, 6.0)]
```

```
In [88]: pd.Series([1, 2, 3, 2, 2, 3]).astype("category")
```

```
Out[88]: 0    1
         1    2
         2    3
         3    2
         4    2
         5    3
        dtype: category
Categories (3, int64): [1, 2, 3]
```

```
In [89]: frame = pd.DataFrame({"col1": ["A", "B", np.nan, "C", "D"], "col2": ["F", np.nan, "G", "H", "I"]})
frame
```

```
Out[89]:   col1  col2
0      A      F
1      B    NaN
2    NaN      G
3      C      H
4      D      I
```

```
In [90]: frame[frame["col2"].isna()]
```

```
Out[90]:   col1  col2
1      B    NaN
```

```
In [91]: frame[frame["col1"].notna()]
```

```
Out[91]:   col1  col2
0      A      F
1      B    NaN
3      C      H
4      D      I
```

```
In [92]: df1 = pd.DataFrame({"key": ["A", "B", "C", "D"], "value": np.random.randn(4)})  
df2 = pd.DataFrame({"key": ["B", "D", "D", "E"], "value": np.random.randn(4)})
```

```
In [93]: pd.merge(df1, df2, on="key")
```

```
Out[93]:   key    value_x    value_y  
0     B -0.335446  1.794026  
1     D  1.224740  1.418379  
2     D  1.224740  0.425891
```

```
In [94]: indexed_df2 = df2.set_index("key")  
pd.merge(df1, indexed_df2, left_on="key", right_index=True)
```

```
Out[94]:   key    value_x    value_y  
1     B -0.335446  1.794026  
3     D  1.224740  1.418379  
3     D  1.224740  0.425891
```

```
In [95]: pd.merge(df1, df2, on="key", how="left")
```

```
Out[95]:   key    value_x    value_y  
0     A  0.429288      NaN  
1     B -0.335446  1.794026  
2     C -0.685751      NaN  
3     D  1.224740  1.418379  
4     D  1.224740  0.425891
```

```
In [96]: pd.merge(df1, df2, on="key", how="right")
```

```
Out[96]:   key    value_x    value_y  
0     B -0.335446  1.794026  
1     D  1.224740  1.418379  
2     D  1.224740  0.425891  
3     E      NaN  0.828731
```

```
In [97]: pd.merge(df1, df2, on="key", how="outer")
```

```
Out[97]:   key  value_x  value_y
```

	key	value_x	value_y
0	A	0.429288	NaN
1	B	-0.335446	1.794026
2	C	-0.685751	NaN
3	D	1.224740	1.418379
4	D	1.224740	0.425891
5	E	NaN	0.828731

```
In [98]: df1 = pd.DataFrame({"city": ["Chicago", "San Francisco", "New York City"], "rank": [1, 2, 3]})  
df2 = pd.DataFrame({"city": ["Chicago", "Boston", "Los Angeles"], "rank": [1, 4, 5]})  
  
pd.concat([df1, df2])
```

```
Out[98]:    city  rank
```

	city	rank
0	Chicago	1
1	San Francisco	2
2	New York City	3
0	Chicago	1
1	Boston	4
2	Los Angeles	5

```
In [99]: pd.concat([df1, df2]).drop_duplicates()
```

```
Out[99]:    city  rank
```

	city	rank
0	Chicago	1
1	San Francisco	2
2	New York City	3
1	Boston	4
2	Los Angeles	5

```
In [100]: df = pd.DataFrame({"x": [1, 3, 5], "y": [2, 4, 6]})  
df
```

```
Out[100]:   x  y
```

	x	y
0	1	2
1	3	4
2	5	6

```
In [101]: firstlast = pd.DataFrame({"String": ["John Smith", "Jane Cook"]})
firstlast["First_Name"] = firstlast["String"].str.split(" ", expand=True)[0]
firstlast["Last_Name"] = firstlast["String"].str.rsplit(" ", expand=True)[1]
firstlast
```

```
Out[101]:      String First_Name Last_Name
0   John Smith        John       Smith
1   Jane Cook         Jane       Cook
```

```
In [102]: firstlast = pd.DataFrame({"string": ["John Smith", "Jane Cook"]})
firstlast["upper"] = firstlast["string"].str.upper()
firstlast["lower"] = firstlast["string"].str.lower()
firstlast["title"] = firstlast["string"].str.title()
firstlast
```

```
Out[102]:      string      upper      lower      title
0   John Smith  JOHN SMITH  john smith  John Smith
1   Jane Cook   JANE COOK  jane cook   Jane Cook
```

```
In [103]: df1 = pd.DataFrame({"key": ["A", "B", "C", "D"], "value": np.random.randn(4)})
df1
```

```
Out[103]:      key      value
0     A    2.710180
1     B   -0.184712
2     C   -0.268376
3     D    1.136070
```

```
In [104]: df2 = pd.DataFrame({"key": ["B", "D", "D", "E"], "value": np.random.randn(4)})
df2
```

```
Out[104]:      key      value
0     B   -1.961649
1     D    0.885771
2     D    0.695118
3     E   -0.265280
```

```
In [105]: inner_join = df1.merge(df2, on=["key"], how="inner")
inner_join
```

```
Out[105]:      key  value_x  value_y
0     B   -0.184712  -1.961649
1     D    1.136070   0.885771
2     D    1.136070   0.695118
```

```
In [106]: left_join = df1.merge(df2, on=["key"], how="left")
left_join
```

```
Out[106]:   key    value_x    value_y
0     A    2.710180      NaN
1     B   -0.184712  -1.961649
2     C   -0.268376      NaN
3     D    1.136070   0.885771
4     D    1.136070   0.695118
```

```
In [107]: right_join = df1.merge(df2, on=["key"], how="right")
right_join
```

```
Out[107]:   key    value_x    value_y
0     B   -0.184712  -1.961649
1     D    1.136070   0.885771
2     D    1.136070   0.695118
3     E      NaN  -0.265280
```

```
In [108]: outer_join = df1.merge(df2, on=["key"], how="outer")
outer_join
```

```
Out[108]:   key    value_x    value_y
0     A    2.710180      NaN
1     B   -0.184712  -1.961649
2     C   -0.268376      NaN
3     D    1.136070   0.885771
4     D    1.136070   0.695118
5     E      NaN  -0.265280
```

```
In [109]: df = pd.DataFrame({"AAA": [1] * 8, "BBB": list(range(0, 8))})  
df
```

```
Out[109]:
```

	AAA	BBB
0	1	0
1	1	1
2	1	2
3	1	3
4	1	4
5	1	5
6	1	6
7	1	7

```
In [110]: series = list(range(1, 5))  
series
```

```
Out[110]: [1, 2, 3, 4]
```

```
In [111]: df.loc[2:5, "AAA"] = series  
df
```

```
Out[111]:
```

	AAA	BBB
0	1	0
1	1	1
2	1	2
3	2	3
4	3	4
5	4	5
6	1	6
7	1	7

```
In [112]: df = pd.DataFrame({  
  
    "class": ["A", "A", "A", "B", "C", "D"],  
    "student_count": [42, 35, 42, 50, 47, 45],  
    "all_pass": ["Yes", "Yes", "Yes", "No", "No", "Yes"]  
  
})  
  
df.drop_duplicates()
```

```
Out[112]:   class  student_count  all_pass  
0          A            42      Yes  
1          A            35      Yes  
3          B            50      No  
4          C            47      No  
5          D            45      Yes
```

```
In [113]: df.drop_duplicates(["class", "student_count"])
```

```
Out[113]:   class  student_count  all_pass  
0          A            42      Yes  
1          A            35      Yes  
3          B            50      No  
4          C            47      No  
5          D            45      Yes
```

```
In [114]: new_row = pd.DataFrame([["E", 51, True]],columns=["class", "student_count", "all_pass"])  
pd.concat([df, new_row])
```

```
Out[114]:   class  student_count  all_pass  
0          A            42      Yes  
1          A            35      Yes  
2          A            42      Yes  
3          B            50      No  
4          C            47      No  
5          D            45      Yes  
0          E            51      True
```

```
In [115]: df = pd.DataFrame({"x": [1, 3, 5], "y": [2, 4, 6]})  
df
```

```
Out[115]:
```

	x	y
0	1	2
1	3	4
2	5	6

```
In [116]: df1 = pd.DataFrame({"key": ["A", "B", "C", "D"], "value": np.random.randn(4)})  
df1
```

```
Out[116]:
```

	key	value
0	A	-1.402688
1	B	-0.545334
2	C	-1.455278
3	D	0.697387

```
In [117]: df2 = pd.DataFrame({"key": ["B", "D", "D", "E"], "value": np.random.randn(4)})  
df2
```

```
Out[117]:
```

	key	value
0	B	-0.665418
1	D	0.008734
2	D	-0.719310
3	E	-0.507211

```
In [118]: inner_join = df1.merge(df2, on=["key"], how="inner")  
inner_join
```

```
Out[118]:
```

	key	value_x	value_y
0	B	-0.545334	-0.665418
1	D	0.697387	0.008734
2	D	0.697387	-0.719310

```
In [119]: left_join = df1.merge(df2, on=["key"], how="left")
left_join
```

```
Out[119]:   key  value_x  value_y
0     A -1.402688      NaN
1     B -0.545334 -0.665418
2     C -1.455278      NaN
3     D  0.697387  0.008734
4     D  0.697387 -0.719310
```

```
In [120]: right_join = df1.merge(df2, on=["key"], how="right")
right_join
```

```
Out[120]:   key  value_x  value_y
0     B -0.545334 -0.665418
1     D  0.697387  0.008734
2     D  0.697387 -0.719310
3     E      NaN -0.507211
```

```
In [121]: outer_join = df1.merge(df2, on=["key"], how="outer")
outer_join
```

```
Out[121]:   key  value_x  value_y
0     A -1.402688      NaN
1     B -0.545334 -0.665418
2     C -1.455278      NaN
3     D  0.697387  0.008734
4     D  0.697387 -0.719310
5     E      NaN -0.507211
```

```
In [122]: outer_join["value_x"] + outer_join["value_y"]
```

```
Out[122]: 0        NaN
1    -1.210752
2        NaN
3     0.706121
4    -0.021924
5        NaN
dtype: float64
```

```
In [123]: outer_join["value_x"].sum()
```

```
Out[123]: -2.008526169978958
```

```
In [124]: outer_join[outer_join["value_x"].isna()]
```

```
Out[124]:   key  value_x  value_y
            5      E      NaN -0.507211
```

```
In [125]: outer_join[outer_join["value_x"].notna()]
```

```
Out[125]:   key  value_x  value_y
            0      A -1.402688      NaN
            1      B -0.545334 -0.665418
            2      C -1.455278      NaN
            3      D  0.697387  0.008734
            4      D  0.697387 -0.719310
```

```
In [126]: outer_join.dropna()
```

```
Out[126]:   key  value_x  value_y
            1      B -0.545334 -0.665418
            3      D  0.697387  0.008734
            4      D  0.697387 -0.719310
```

```
In [127]: outer_join.fillna(method="ffill")
```

```
Out[127]:   key  value_x  value_y
            0      A -1.402688      NaN
            1      B -0.545334 -0.665418
            2      C -1.455278 -0.665418
            3      D  0.697387  0.008734
            4      D  0.697387 -0.719310
            5      E  0.697387 -0.507211
```

```
In [128]: outer_join["value_x"].fillna(outer_join["value_x"].mean())
```

```
Out[128]: 0    -1.402688
           1    -0.545334
           2    -1.455278
           3     0.697387
           4     0.697387
           5    -0.401705
Name: value_x, dtype: float64
```

```
In [129]: s = pd.Series([1, 3, 5, np.nan, 6, 8])
s
```

```
Out[129]: 0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

```
In [130]: dates = pd.date_range("20130101", periods=6)
dates
```

```
Out[130]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                           '2013-01-05', '2013-01-06'],
                           dtype='datetime64[ns]', freq='D')
```

```
In [131]: df = pd.DataFrame(np.random.randn(6, 4), index=dates, columns=list("ABCD"))
df
```

```
Out[131]:
```

	A	B	C	D
2013-01-01	-0.591099	-1.749225	1.031762	0.972904
2013-01-02	0.437900	-0.154981	0.621125	0.812640
2013-01-03	-0.277972	0.613546	-1.647452	0.395107
2013-01-04	-0.232215	1.631459	0.539724	-0.872221
2013-01-05	0.427207	-0.571715	1.176223	0.509280
2013-01-06	-0.876173	1.281415	0.015445	-2.132941

```
In [132]: df2 = pd.DataFrame({
    "A": 1.0,
    "B": pd.Timestamp("20130102"),
    "C": pd.Series(1, index=list(range(4)), dtype="float32"),
    "D": np.array([3] * 4, dtype="int32"),
    "E": pd.Categorical(["test", "train", "test", "train"]),
    "F": "foo"
})
```

```
df2
```

```
Out[132]:
```

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

```
In [133]: df2.index
```

```
Out[133]: Int64Index([0, 1, 2, 3], dtype='int64')
```

```
In [134]: df.to_numpy()
```

```
Out[134]: array([[-0.59109898, -1.749225 ,  1.03176168,  0.97290379],
                  [ 0.43790005, -0.15498061,  0.62112534,  0.81264041],
                  [-0.27797154,  0.6135462 , -1.64745161,  0.39510719],
                  [-0.23221497,  1.63145916,  0.53972385, -0.87222052],
                  [ 0.42720723, -0.57171521,  1.17622299,  0.50928032],
                  [-0.87617273,  1.28141542,  0.01544495, -2.13294079]])
```

```
In [135]: df2.to_numpy()
```

```
Out[135]: array([[1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
                  [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo'],
                  [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
                  [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo']],
                 dtype=object)
```

```
In [136]: df.sort_index(axis=1, ascending=False)
```

```
Out[136]:
```

	D	C	B	A
2013-01-01	0.972904	1.031762	-1.749225	-0.591099
2013-01-02	0.812640	0.621125	-0.154981	0.437900
2013-01-03	0.395107	-1.647452	0.613546	-0.277972
2013-01-04	-0.872221	0.539724	1.631459	-0.232215
2013-01-05	0.509280	1.176223	-0.571715	0.427207
2013-01-06	-2.132941	0.015445	1.281415	-0.876173

```
In [137]: df.sort_values(by="B")
```

```
Out[137]:
```

	A	B	C	D
2013-01-01	-0.591099	-1.749225	1.031762	0.972904
2013-01-05	0.427207	-0.571715	1.176223	0.509280
2013-01-02	0.437900	-0.154981	0.621125	0.812640
2013-01-03	-0.277972	0.613546	-1.647452	0.395107
2013-01-06	-0.876173	1.281415	0.015445	-2.132941
2013-01-04	-0.232215	1.631459	0.539724	-0.872221

```
In [138]: df[0:3]
```

```
Out[138]:
```

	A	B	C	D
2013-01-01	-0.591099	-1.749225	1.031762	0.972904
2013-01-02	0.437900	-0.154981	0.621125	0.812640
2013-01-03	-0.277972	0.613546	-1.647452	0.395107

```
In [139]: df["20130102":"20130104"]
```

```
Out[139]:
```

	A	B	C	D
2013-01-02	0.437900	-0.154981	0.621125	0.812640
2013-01-03	-0.277972	0.613546	-1.647452	0.395107
2013-01-04	-0.232215	1.631459	0.539724	-0.872221

```
In [140]: df.loc[dates[0]]
```

```
Out[140]: A    -0.591099  
B    -1.749225  
C    1.031762  
D    0.972904  
Name: 2013-01-01 00:00:00, dtype: float64
```

```
In [141]: df.loc[:, ["A", "B"]]
```

```
Out[141]:
```

	A	B
2013-01-01	-0.591099	-1.749225
2013-01-02	0.437900	-0.154981
2013-01-03	-0.277972	0.613546
2013-01-04	-0.232215	1.631459
2013-01-05	0.427207	-0.571715
2013-01-06	-0.876173	1.281415

```
In [142]: df.loc["20130102":"20130104", ["A", "B"]]
```

```
Out[142]:
```

	A	B
2013-01-02	0.437900	-0.154981
2013-01-03	-0.277972	0.613546
2013-01-04	-0.232215	1.631459

```
In [143]: df.loc["20130102", ["A", "B"]]
```

```
Out[143]: A    0.437900  
B    -0.154981  
Name: 2013-01-02 00:00:00, dtype: float64
```

```
In [144]: df.at[dates[0], "A"]
```

```
Out[144]: -0.5910989777117921
```

```
In [145]: df.iloc[3]
```

```
Out[145]: A    -0.232215  
B     1.631459  
C     0.539724  
D    -0.872221  
Name: 2013-01-04 00:00:00, dtype: float64
```

```
In [146]: df.iloc[3:5, 0:2]
```

```
Out[146]:
```

	A	B
2013-01-04	-0.232215	1.631459
2013-01-05	0.427207	-0.571715

```
In [147]: df.iloc[[1, 2, 4], [0, 2]]
```

```
Out[147]:
```

	A	C
2013-01-02	0.437900	0.621125
2013-01-03	-0.277972	-1.647452
2013-01-05	0.427207	1.176223

```
In [148]: df.iloc[1:3, :]
```

```
Out[148]:
```

	A	B	C	D
2013-01-02	0.437900	-0.154981	0.621125	0.812640
2013-01-03	-0.277972	0.613546	-1.647452	0.395107

```
In [149]: df.iloc[:, 1:3]
```

```
Out[149]:
```

	B	C
2013-01-01	-1.749225	1.031762
2013-01-02	-0.154981	0.621125
2013-01-03	0.613546	-1.647452
2013-01-04	1.631459	0.539724
2013-01-05	-0.571715	1.176223
2013-01-06	1.281415	0.015445

```
In [150]: df.iloc[1, 1]
```

```
Out[150]: -0.15498061171987595
```

```
In [151]: df[df["A"] > 0]
```

```
Out[151]:
```

	A	B	C	D
2013-01-02	0.437900	-0.154981	0.621125	0.81264
2013-01-05	0.427207	-0.571715	1.176223	0.50928

```
In [152]: df[df > 0]
```

```
Out[152]:
```

	A	B	C	D
2013-01-01	NaN	NaN	1.031762	0.972904
2013-01-02	0.437900	NaN	0.621125	0.812640
2013-01-03	NaN	0.613546	NaN	0.395107
2013-01-04	NaN	1.631459	0.539724	NaN
2013-01-05	0.427207	NaN	1.176223	0.509280
2013-01-06	NaN	1.281415	0.015445	NaN

```
In [153]: df2 = df.copy()
df2["E"] = ["one", "one", "two", "three", "four", "three"]
df2
```

```
Out[153]:
```

	A	B	C	D	E
2013-01-01	-0.591099	-1.749225	1.031762	0.972904	one
2013-01-02	0.437900	-0.154981	0.621125	0.812640	one
2013-01-03	-0.277972	0.613546	-1.647452	0.395107	two
2013-01-04	-0.232215	1.631459	0.539724	-0.872221	three
2013-01-05	0.427207	-0.571715	1.176223	0.509280	four
2013-01-06	-0.876173	1.281415	0.015445	-2.132941	three

```
In [154]: df2[df2["E"].isin(["two", "four"])]
```

```
Out[154]:
```

	A	B	C	D	E
2013-01-03	-0.277972	0.613546	-1.647452	0.395107	two
2013-01-05	0.427207	-0.571715	1.176223	0.509280	four

In [155]:

```
s1 = pd.Series([1, 2, 3, 4, 5, 6], index=pd.date_range("20130102", periods=6))
s1
```

Out[155]:

2013-01-02	1
2013-01-03	2
2013-01-04	3
2013-01-05	4
2013-01-06	5
2013-01-07	6

Freq: D, dtype: int64

In [156]:

```
df.at[dates[0], "A"] = 0
```

In [157]:

```
df.iat[0, 1] = 0
```

In [158]:

```
df.loc[:, "D"] = np.array([5] * len(df))
df
```

```
C:\Users\pytho\AppData\Local\Temp\ipykernel_8580\568071402.py:1: DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt to set the values inplace instead of always setting a new array. To retain the old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-unique, `df.iisetitem(i, newvals)`
df.loc[:, "D"] = np.array([5] * len(df))
```

Out[158]:

	A	B	C	D
2013-01-01	0.000000	0.000000	1.031762	5
2013-01-02	0.437900	-0.154981	0.621125	5
2013-01-03	-0.277972	0.613546	-1.647452	5
2013-01-04	-0.232215	1.631459	0.539724	5
2013-01-05	0.427207	-0.571715	1.176223	5
2013-01-06	-0.876173	1.281415	0.015445	5

In [159]:

```
df2 = df.copy()
df2[df2 > 0] = -df2
df2
```

Out[159]:

	A	B	C	D
2013-01-01	0.000000	0.000000	-1.031762	-5
2013-01-02	-0.437900	-0.154981	-0.621125	-5
2013-01-03	-0.277972	-0.613546	-1.647452	-5
2013-01-04	-0.232215	-1.631459	-0.539724	-5
2013-01-05	-0.427207	-0.571715	-1.176223	-5
2013-01-06	-0.876173	-1.281415	-0.015445	-5

```
In [160]: df1 = df.reindex(index=dates[0:4], columns=list(df.columns) + ["E"])
df1.loc[dates[0] : dates[1], "E"] = 1
df1
```

```
Out[160]:
```

	A	B	C	D	E
2013-01-01	0.000000	0.000000	1.031762	5	1.0
2013-01-02	0.437900	-0.154981	0.621125	5	1.0
2013-01-03	-0.277972	0.613546	-1.647452	5	NaN
2013-01-04	-0.232215	1.631459	0.539724	5	NaN

```
In [161]: df1.dropna(how="any")
```

```
Out[161]:
```

	A	B	C	D	E
2013-01-01	0.0000	0.000000	1.031762	5	1.0
2013-01-02	0.4379	-0.154981	0.621125	5	1.0

```
In [162]: df1.fillna(value=5)
```

```
Out[162]:
```

	A	B	C	D	E
2013-01-01	0.000000	0.000000	1.031762	5	1.0
2013-01-02	0.437900	-0.154981	0.621125	5	1.0
2013-01-03	-0.277972	0.613546	-1.647452	5	5.0
2013-01-04	-0.232215	1.631459	0.539724	5	5.0

```
In [163]: pd.isna(df1)
```

```
Out[163]:
```

	A	B	C	D	E
2013-01-01	False	False	False	False	False
2013-01-02	False	False	False	False	False
2013-01-03	False	False	False	False	True
2013-01-04	False	False	False	False	True

```
In [164]: df.mean()
```

```
Out[164]: A    -0.086875
B     0.466621
C     0.289471
D     5.000000
dtype: float64
```

```
In [165]: df.mean(1)
```

```
Out[165]: 2013-01-01    1.507940
2013-01-02    1.476011
2013-01-03    0.922031
2013-01-04    1.734742
2013-01-05    1.507929
2013-01-06    1.355172
Freq: D, dtype: float64
```

```
In [166]: s = pd.Series([1, 3, 5, np.nan, 6, 8], index=dates).shift(2)
s
```

```
Out[166]: 2013-01-01    NaN
2013-01-02    NaN
2013-01-03    1.0
2013-01-04    3.0
2013-01-05    5.0
2013-01-06    NaN
Freq: D, dtype: float64
```

```
In [167]: df.sub(s, axis="index")
```

```
Out[167]:
```

	A	B	C	D
2013-01-01	NaN	NaN	NaN	NaN
2013-01-02	NaN	NaN	NaN	NaN
2013-01-03	-1.277972	-0.386454	-2.647452	4.0
2013-01-04	-3.232215	-1.368541	-2.460276	2.0
2013-01-05	-4.572793	-5.571715	-3.823777	0.0
2013-01-06	NaN	NaN	NaN	NaN

```
In [168]: df.apply(np.cumsum)
```

```
Out[168]:
```

	A	B	C	D
2013-01-01	0.000000	0.000000	1.031762	5
2013-01-02	0.437900	-0.154981	1.652887	10
2013-01-03	0.159929	0.458566	0.005435	15
2013-01-04	-0.072286	2.090025	0.545159	20
2013-01-05	0.354921	1.518310	1.721382	25
2013-01-06	-0.521252	2.799725	1.736827	30

```
In [169]: df.apply(lambda x: x.max() - x.min())
```

```
Out[169]: A    1.314073  
          B    2.203174  
          C    2.823675  
          D    0.000000  
         dtype: float64
```

```
In [1]: import pandas as pd  
import numpy as np
```

```
In [2]: s = pd.Series(np.random.randint(0, 7, size=10))  
s
```

```
Out[2]: 0      0  
1      6  
2      5  
3      6  
4      5  
5      5  
6      2  
7      2  
8      4  
9      2  
dtype: int32
```

```
In [3]: s.value_counts()
```

```
Out[3]: 5      3  
2      3  
6      2  
0      1  
4      1  
dtype: int64
```

```
In [4]: s = pd.Series(["A", "B", "C", "Aaba", "Baca", np.nan, "CABA", "dog", "cat"])  
s.str.lower()
```

```
Out[4]: 0      a  
1      b  
2      c  
3     aaba  
4     baca  
5    NaN  
6     caba  
7     dog  
8     cat  
dtype: object
```

```
In [5]: df = pd.DataFrame(np.random.randn(10, 4))  
df
```

```
Out[5]:
```

	0	1	2	3
0	0.423742	0.285896	-1.319679	0.106474
1	-0.917753	-0.612802	0.510887	-0.192883
2	-0.398895	1.744283	0.392814	1.252180
3	1.275378	-2.030125	1.377954	0.047816
4	0.541705	0.955300	-0.510238	0.690620
5	-0.968414	-0.495862	1.229128	0.968220
6	-1.478773	0.418985	1.010736	-1.494321
7	0.743932	-0.563562	0.986714	0.625697
8	-0.407228	-1.016760	0.617824	-0.370512
9	-0.129449	0.430960	0.192333	0.270365

```
In [6]: pieces = [df[:3], df[3:7], df[7:]]  
pd.concat(pieces)
```

```
Out[6]:
```

	0	1	2	3
0	0.423742	0.285896	-1.319679	0.106474
1	-0.917753	-0.612802	0.510887	-0.192883
2	-0.398895	1.744283	0.392814	1.252180
3	1.275378	-2.030125	1.377954	0.047816
4	0.541705	0.955300	-0.510238	0.690620
5	-0.968414	-0.495862	1.229128	0.968220
6	-1.478773	0.418985	1.010736	-1.494321
7	0.743932	-0.563562	0.986714	0.625697
8	-0.407228	-1.016760	0.617824	-0.370512
9	-0.129449	0.430960	0.192333	0.270365

```
In [9]: left = pd.DataFrame({"key": ["foo", "foo"], "lval": [1, 2]})  
right = pd.DataFrame({"key": ["foo", "foo"], "rval": [4, 5]})  
left
```

```
Out[9]:
```

	key	lval
0	foo	1
1	foo	2

```
In [10]: pd.merge(left, right, on="key")
```

```
Out[10]:
```

	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

```
In [11]: left = pd.DataFrame({"key": ["foo", "bar"], "lval": [1, 2]})  
right = pd.DataFrame({"key": ["foo", "bar"], "rval": [4, 5]})  
left
```

```
Out[11]:
```

	key	lval
0	foo	1
1	bar	2

```
In [12]: pd.merge(left, right, on="key")
```

```
Out[12]:
```

	key	lval	rval
0	foo	1	4
1	bar	2	5

```
In [13]: df = pd.DataFrame({
```

```
    "A": ["foo", "bar", "foo", "bar", "foo", "bar", "foo", "foo"],  
    "B": ["one", "one", "two", "three", "two", "two", "one", "three"],  
    "C": np.random.randn(8),  
    "D": np.random.randn(8)
```

```
})  
df
```

```
Out[13]:
```

	A	B	C	D
0	foo	one	-0.663656	-0.382136
1	bar	one	-1.314220	-1.048106
2	foo	two	0.391064	-0.693546
3	bar	three	-0.778172	0.695501
4	foo	two	1.171063	-0.340544
5	bar	two	-0.688497	0.840714
6	foo	one	-0.762570	-0.991745
7	foo	three	-1.449696	0.397626

```
In [14]: df.groupby("A").sum()
```

```
Out[14]:      C      D
```

	A	
bar	-2.780889	0.488108
foo	-1.313795	-2.010344

```
In [15]: df.groupby(["A", "B"]).sum()
```

```
Out[15]:      C      D
```

	A	B	
	one	-1.314220	-1.048106
bar	three	-0.778172	0.695501
	two	-0.688497	0.840714
	one	-1.426226	-1.373881
foo	three	-1.449696	0.397626
	two	1.562128	-1.034089

```
In [16]: tuples = list(zip(*[
    ["bar", "bar", "baz", "baz", "foo", "foo", "qux", "qux"],
    ["one", "two", "one", "two", "one", "two", "one", "two"]
]))
index = pd.MultiIndex.from_tuples(tuples, names=["first", "second"])
df = pd.DataFrame(np.random.randn(8, 2), index=index, columns=["A", "B"])
df2 = df[:4]
df2
```

```
Out[16]:      A      B
```

	first	second	
bar	one	-2.314275	-0.404936
	two	0.370079	-0.128071
baz	one	1.472769	-1.160009
	two	-1.060644	-1.309345

```
In [17]: stacked = df2.stack()
stacked
```

```
Out[17]: first    second
         bar      one      A   -2.314275
                      B   -0.404936
                     two     A    0.370079
                      B   -0.128071
        baz      one      A    1.472769
                      B   -1.160009
                     two     A   -1.060644
                      B   -1.309345
dtype: float64
```

```
In [18]: stacked.unstack()
```

```
Out[18]:          A          B
first  second
bar
       one   -2.314275  -0.404936
       two    0.370079  -0.128071
baz
       one    1.472769  -1.160009
       two   -1.060644  -1.309345
```

```
In [19]: stacked.unstack(1)
```

```
Out[19]:      second      one      two
first
bar
       A   -2.314275  0.370079
       B   -0.404936  -0.128071
baz
       A    1.472769  -1.060644
       B   -1.160009  -1.309345
```

```
In [20]: df = pd.DataFrame({
    "A": ["one", "one", "two", "three"] * 3,
    "B": ["A", "B", "C"] * 4,
    "C": ["foo", "foo", "foo", "bar", "bar", "bar"] * 2,
    "D": np.random.randn(12),
    "E": np.random.randn(12)
})
df
```

Out[20]:

	A	B	C	D	E
0	one	A	foo	-1.156964	1.545813
1	one	B	foo	-1.975856	-0.172538
2	two	C	foo	-1.677984	-1.310321
3	three	A	bar	-1.159864	-0.761936
4	one	B	bar	0.408105	1.079825
5	one	C	bar	-0.717914	-1.394031
6	two	A	foo	0.620242	-0.099728
7	three	B	foo	0.315321	-0.546775
8	one	C	foo	-0.243819	1.283591
9	one	A	bar	0.718317	-0.877277
10	two	B	bar	0.747791	-0.748307
11	three	C	bar	-0.147173	0.632441

```
In [21]: pd.pivot_table(df, values="D", index=["A", "B"], columns=["C"])
```

Out[21]:

	C		bar	foo
	A	B		
	A		0.718317	-1.156964
	one	B	0.408105	-1.975856
		C	-0.717914	-0.243819
		A	-1.159864	NaN
	three	B	NaN	0.315321
		C	-0.147173	NaN
		A	NaN	0.620242
	two	B	0.747791	NaN
		C	NaN	-1.677984

```
In [22]: rng = pd.date_range("1/1/2012", periods=100, freq="S")
ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
ts.resample("5Min").sum()
```

```
Out[22]: 2012-01-01    26075
          Freq: 5T, dtype: int32
```

```
In [23]: rng = pd.date_range("3/6/2012 00:00", periods=5, freq="D")
ts = pd.Series(np.random.randn(len(rng)), rng)
ts
```

```
Out[23]: 2012-03-06    0.023926
          2012-03-07   -0.602996
          2012-03-08    0.686197
          2012-03-09    0.535357
          2012-03-10   -1.408127
          Freq: D, dtype: float64
```

```
In [24]: ts_utc = ts.tz_localize("UTC")
ts_utc
```

```
Out[24]: 2012-03-06 00:00:00+00:00    0.023926
          2012-03-07 00:00:00+00:00   -0.602996
          2012-03-08 00:00:00+00:00    0.686197
          2012-03-09 00:00:00+00:00    0.535357
          2012-03-10 00:00:00+00:00   -1.408127
          Freq: D, dtype: float64
```

```
In [25]: rng = pd.date_range("1/1/2012", periods=5, freq="M")
ts = pd.Series(np.random.randn(len(rng)), index=rng)
ts
```

```
Out[25]: 2012-01-31   -1.062234
          2012-02-29    0.942182
          2012-03-31   -0.908925
          2012-04-30    0.171292
          2012-05-31   -2.773022
          Freq: M, dtype: float64
```

```
In [26]: ps = ts.to_period()
ps
```

```
Out[26]: 2012-01   -1.062234
          2012-02    0.942182
          2012-03   -0.908925
          2012-04    0.171292
          2012-05   -2.773022
          Freq: M, dtype: float64
```

```
In [27]: ps.to_timestamp()
```

```
Out[27]: 2012-01-01    -1.062234
2012-02-01     0.942182
2012-03-01    -0.908925
2012-04-01     0.171292
2012-05-01    -2.773022
Freq: MS, dtype: float64
```

```
In [28]: prng = pd.period_range("1990Q1", "2000Q4", freq="Q-NOV")
ts = pd.Series(np.random.randn(len(prng)), prng)
ts.index = (prng.asfreq("M", "e") + 1).asfreq("H", "s") + 9
ts.head()
```

```
Out[28]: 1990-03-01 09:00    -0.745830
1990-06-01 09:00    -0.117445
1990-09-01 09:00    -0.189264
1990-12-01 09:00     0.541704
1991-03-01 09:00    -0.280971
Freq: H, dtype: float64
```

```
In [29]: df = pd.DataFrame({"id": [1, 2, 3, 4, 5, 6], "raw_grade": ["a", "b", "b", "a", "a", "a"]})
df.head()
```

```
Out[29]:   id  raw_grade
0   1          a
1   2          b
2   3          b
3   4          a
4   5          a
```

```
In [30]: df["grade"] = df["raw_grade"].astype("category")
In [125]: df["grade"]
```

```
Out[30]: 0    a
1    b
2    b
3    a
4    a
5    e
Name: grade, dtype: category
Categories (3, object): ['a', 'b', 'e']
```

```
In [31]: df["grade"].cat.categories = ["very good", "good", "very bad"]
```

```
In [32]: df["grade"] = df["grade"].cat.set_categories(["very bad", "bad", "medium", "good"])
df["grade"]
```

```
Out[32]: 0    very good
1        good
2        good
3    very good
4    very good
5    very bad
Name: grade, dtype: category
Categories (5, object): ['very bad', 'bad', 'medium', 'good', 'very good']
```

```
In [33]: df.sort_values(by="grade")
```

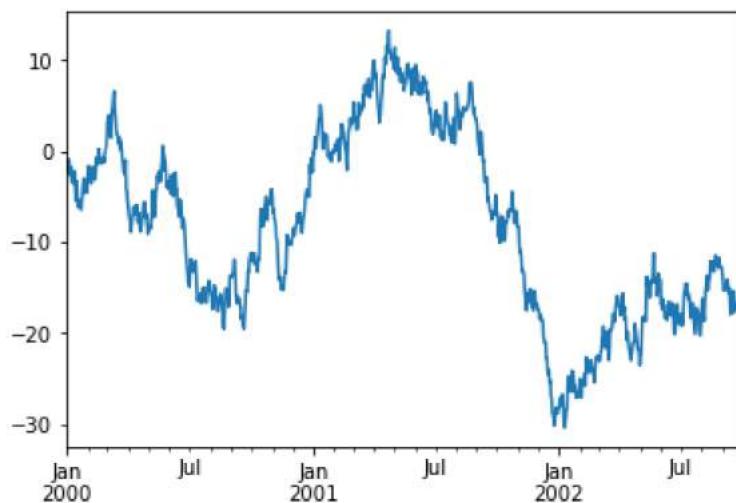
```
Out[33]:   id  raw_grade     grade
      5       6          e  very bad
      1       2          b      good
      2       3          b      good
      0       1          a  very good
      3       4          a  very good
      4       5          a  very good
```

```
In [34]: df.groupby("grade").size()
```

```
Out[34]: grade
very bad      1
bad           0
medium         0
good          2
very good     3
dtype: int64
```

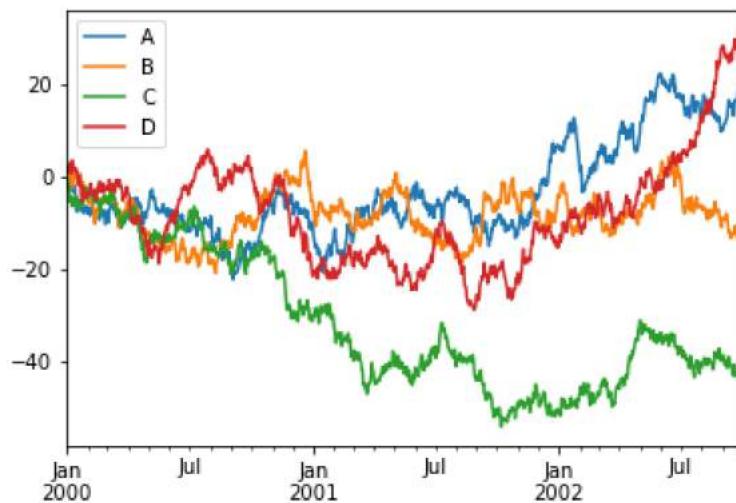
```
In [35]: import matplotlib.pyplot as plt
plt.close("all")
```

```
In [36]: ts = pd.Series(np.random.randn(1000), index=pd.date_range("1/1/2000", periods=1000))
ts = ts.cumsum()
ts.plot();
```



```
In [37]: df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=["A", "B", "C", "D"])
df = df.cumsum()
plt.figure();
df.plot();
plt.legend(loc='best');
```

<Figure size 432x288 with 0 Axes>



```
In [40]: df.to_csv("foo.csv")
pd.read_csv("foo.csv")
```

Out[40]:

	Unnamed: 0	A	B	C	D
0	2000-01-01	0.070171	-0.384287	-0.333774	1.929496
1	2000-01-02	0.366578	-0.826180	0.025036	1.947304
2	2000-01-03	0.031841	-2.156061	-0.295405	3.846140
3	2000-01-04	-1.453527	-1.720770	0.681525	4.016645
4	2000-01-05	-0.506616	-1.748306	-0.953566	4.466979
...
995	2002-09-22	-8.952960	12.174642	-44.968475	21.234458
996	2002-09-23	-9.990733	13.157426	-43.955102	22.196782
997	2002-09-24	-10.370856	13.446153	-44.753827	23.111634
998	2002-09-25	-10.254712	13.043342	-44.844481	22.830099
999	2002-09-26	-9.240218	13.054899	-44.254345	22.267607

1000 rows × 5 columns

```
In [41]: df.to_hdf("foo.h5", "df")
pd.read_hdf("foo.h5", "df")
```

Out[41]:

	A	B	C	D
2000-01-01	0.070171	-0.384287	-0.333774	1.929496
2000-01-02	0.366578	-0.826180	0.025036	1.947304
2000-01-03	0.031841	-2.156061	-0.295405	3.846140
2000-01-04	-1.453527	-1.720770	0.681525	4.016645
2000-01-05	-0.506616	-1.748306	-0.953566	4.466979
...
2002-09-22	-8.952960	12.174642	-44.968475	21.234458
2002-09-23	-9.990733	13.157426	-43.955102	22.196782
2002-09-24	-10.370856	13.446153	-44.753827	23.111634
2002-09-25	-10.254712	13.043342	-44.844481	22.830099
2002-09-26	-9.240218	13.054899	-44.254345	22.267607

1000 rows × 4 columns

```
In [42]: df.to_excel("foo.xlsx", sheet_name="Sheet1")
pd.read_excel("foo.xlsx", "Sheet1", index_col=None, na_values=["NA"])
```

Out[42]:

	Unnamed: 0	A	B	C	D
0	2000-01-01	0.070171	-0.384287	-0.333774	1.929496
1	2000-01-02	0.366578	-0.826180	0.025036	1.947304
2	2000-01-03	0.031841	-2.156061	-0.295405	3.846140
3	2000-01-04	-1.453527	-1.720770	0.681525	4.016645
4	2000-01-05	-0.506616	-1.748306	-0.953566	4.466979
...
995	2002-09-22	-8.952960	12.174642	-44.968475	21.234458
996	2002-09-23	-9.990733	13.157426	-43.955102	22.196782
997	2002-09-24	-10.370856	13.446153	-44.753827	23.111634
998	2002-09-25	-10.254712	13.043342	-44.844481	22.830099
999	2002-09-26	-9.240218	13.054899	-44.254345	22.267607

1000 rows × 5 columns

```
In [43]: s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
s
```

```
Out[43]: a    -0.168186
         b    -2.300513
         c    -1.303243
         d     1.538910
         e    -1.602989
dtype: float64
```

```
In [44]: d = {"b": 1, "a": 0, "c": 2}
pd.Series(d)
```

```
Out[44]: b    1
         a    0
         c    2
dtype: int64
```

```
In [45]: d = {"a": 0.0, "b": 1.0, "c": 2.0}
pd.Series(d)
```

```
Out[45]: a    0.0
         b    1.0
         c    2.0
dtype: float64
```

```
In [46]: pd.Series(d, index=["b", "c", "d", "a"])
```

```
Out[46]: b    1.0
          c    2.0
          d    NaN
          a    0.0
          dtype: float64
```

```
In [47]: pd.Series(5.0, index=["a", "b", "c", "d", "e"])
```

```
Out[47]: a    5.0
          b    5.0
          c    5.0
          d    5.0
          e    5.0
          dtype: float64
```

```
In [48]: s[0]
```

```
Out[48]: -0.1681860503643496
```

```
In [49]: s[:3]
```

```
Out[49]: a    -0.168186
          b    -2.300513
          c    -1.303243
          dtype: float64
```

```
In [50]: s[s > s.median()]
```

```
Out[50]: a    -0.168186
          d    1.538910
          dtype: float64
```

```
In [51]: s[[4, 3, 1]]
```

```
Out[51]: e    -1.602989
          d    1.538910
          b    -2.300513
          dtype: float64
```

```
In [52]: np.exp(s)
```

```
Out[52]: a    0.845197
          b    0.100207
          c    0.271649
          d    4.659509
          e    0.201294
          dtype: float64
```

```
In [53]: s.array
```

```
Out[53]: <PandasArray>
[-0.1681860503643496, -2.3005130449484565, -1.3032428199311967,
 1.5389100807195653, -1.6029891733694224]
Length: 5, dtype: float64
```

```
In [54]: s.to_numpy()
```

```
Out[54]: array([-0.16818605, -2.30051304, -1.30324282,  1.53891008, -1.60298917])
```

```
In [55]: s["a"]
```

```
Out[55]: -0.1681860503643496
```

```
In [56]: s["e"] = 12.0
s
```

```
Out[56]: a    -0.168186
         b    -2.300513
         c    -1.303243
         d     1.538910
         e    12.000000
dtype: float64
```

```
In [57]: np.exp(s)
```

```
Out[57]: a      0.845197
         b      0.100207
         c      0.271649
         d      4.659509
         e  162754.791419
dtype: float64
```

```
In [58]: s[1:] + s[:-1]
```

```
Out[58]: a      NaN
         b     -4.601026
         c     -2.606486
         d      3.077820
         e      NaN
dtype: float64
```

```
In [59]: s = pd.Series(np.random.randn(5), name="something")
s
```

```
Out[59]: 0   -0.522842
         1    0.741873
         2   -1.460176
         3   -0.526032
         4   -0.180085
Name: something, dtype: float64
```

```
In [60]: s2 = s.rename("different")
s2.name
```

```
Out[60]: 'different'
```

```
In [61]: d = {
    "one": pd.Series([1.0, 2.0, 3.0], index=["a", "b", "c"]),
    "two": pd.Series([1.0, 2.0, 3.0, 4.0], index=["a", "b", "c", "d"]),
}

df = pd.DataFrame(d)
df
```

```
Out[61]:   one  two
a    1.0  1.0
b    2.0  2.0
c    3.0  3.0
d    NaN  4.0
```

```
In [62]: pd.DataFrame(d, index=["d", "b", "a"])
```

```
Out[62]:   one  two
d    NaN  4.0
b    2.0  2.0
a    1.0  1.0
```

```
In [63]: pd.DataFrame(d, index=["d", "b", "a"], columns=["two", "three"])
```

```
Out[63]:   two  three
d    4.0   NaN
b    2.0   NaN
a    1.0   NaN
```

```
In [64]: df.index
```

```
Out[64]: Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
In [65]: df.columns
```

```
Out[65]: Index(['one', 'two'], dtype='object')
```

```
In [66]: d = {"one": [1.0, 2.0, 3.0, 4.0], "two": [4.0, 3.0, 2.0, 1.0]}
```

```
Out[66]:
```

	one	two
0	1.0	4.0
1	2.0	3.0
2	3.0	2.0
3	4.0	1.0

```
In [67]: pd.DataFrame(d, index=["a", "b", "c", "d"])
```

```
Out[67]:
```

	one	two
a	1.0	4.0
b	2.0	3.0
c	3.0	2.0
d	4.0	1.0

```
In [68]: data = np.zeros((2,), dtype=[("A", "i4"), ("B", "f4"), ("C", "a10")])
data[:] = [(1, 2.0, "Hello"), (2, 3.0, "World")]
pd.DataFrame(data)
```

```
Out[68]:
```

	A	B	C
0	1	2.0	b'Hello'
1	2	3.0	b'World'

```
In [69]: pd.DataFrame(data, index=["first", "second"])
```

```
Out[69]:
```

	A	B	C
first	1	2.0	b'Hello'
second	2	3.0	b'World'

```
In [70]: pd.DataFrame(data, columns=["C", "A", "B"])
```

```
Out[70]:
```

	C	A	B
0	b'Hello'	1	2.0
1	b'World'	2	3.0

```
In [71]: data2 = [{"a": 1, "b": 2}, {"a": 5, "b": 10, "c": 20}]
pd.DataFrame(data2)
```

```
Out[71]:      a   b   c
              0   1   2   NaN
              1   5  10  20.0
```

```
In [72]: pd.DataFrame(data2, index=["first", "second"])
```

```
Out[72]:      a   b   c
first     1   2   NaN
second    5  10  20.0
```

```
In [73]: pd.DataFrame(data2, columns=["a", "b"])
```

```
Out[73]:      a   b
              0   1   2
              1   5  10
```

```
In [74]: pd.DataFrame({
```

```
    ("a", "b"): {("A", "B"): 1, ("A", "C"): 2},
    ("a", "a"): {("A", "C"): 3, ("A", "B"): 4},
    ("a", "c"): {("A", "B"): 5, ("A", "C"): 6},
    ("b", "a"): {("A", "C"): 7, ("A", "B"): 8},
    ("b", "b"): {("A", "D"): 9, ("A", "B"): 10}
```

```
})
```

```
Out[74]:      a          b
              b   a   c   a   b
              B   1.0  4.0  5.0  8.0  10.0
              A   C   2.0  3.0  6.0  7.0   NaN
              D   NaN  NaN  NaN  NaN   9.0
```

```
In [75]: from collections import namedtuple
Point = namedtuple("Point", "x y")
pd.DataFrame([Point(0, 0), Point(0, 3), (2, 3)])
```

```
Out[75]:      x   y
              0   0   0
              1   0   3
              2   2   3
```

```
In [76]: Point3D = namedtuple("Point3D", "x y z")
```

```
In [77]: pd.DataFrame([Point3D(0, 0, 0), Point3D(0, 3, 5), Point(2, 3)])
```

```
Out[77]:
```

	x	y	z
0	0	0	0.0
1	0	3	5.0
2	2	3	NaN

```
In [78]: from dataclasses import make_dataclass  
Point = make_dataclass("Point", [("x", int), ("y", int)])  
pd.DataFrame([Point(0, 0), Point(0, 3), Point(2, 3)])
```

```
Out[78]:
```

	x	y
0	0	0
1	0	3
2	2	3

```
In [79]: pd.DataFrame.from_dict(dict([("A", [1, 2, 3]), ("B", [4, 5, 6])]))
```

```
Out[79]:
```

	A	B
0	1	4
1	2	5
2	3	6

```
In [80]: pd.DataFrame.from_dict(  
dict([("A", [1, 2, 3]), ("B", [4, 5, 6])]),  
orient="index",  
columns=["one", "two", "three"],  
)
```

```
Out[80]:
```

	one	two	three
A	1	2	3
B	4	5	6

```
In [81]: pd.DataFrame.from_records(data, index="C")
```

```
Out[81]:
```

	A	B
C		
b'Hello'	1	2.0
b'World'	2	3.0

```
In [82]: df["three"] = df["one"] * df["two"]
df["flag"] = df["one"] > 2
df
```

```
Out[82]:   one  two  three  flag
          a    1.0  1.0   1.0  False
          b    2.0  2.0   4.0  False
          c    3.0  3.0   9.0   True
          d    NaN  4.0   NaN  False
```

```
In [83]: del df["two"]
three = df.pop("three")
df
```

```
Out[83]:   one  flag
          a    1.0  False
          b    2.0  False
          c    3.0   True
          d    NaN  False
```

```
In [84]: df["foo"] = "bar"
df
```

```
Out[84]:   one  flag  foo
          a    1.0  False  bar
          b    2.0  False  bar
          c    3.0   True  bar
          d    NaN  False  bar
```

```
In [85]: df["one_trunc"] = df["one"][:2]
df
```

```
Out[85]:   one  flag  foo  one_trunc
          a    1.0  False  bar       1.0
          b    2.0  False  bar       2.0
          c    3.0   True  bar      NaN
          d    NaN  False  bar      NaN
```

```
In [86]: df.insert(1, "bar", df["one"])
df
```

```
Out[86]:   one  bar  flag  foo  one_trunc
```

a	1.0	1.0	False	bar	1.0
b	2.0	2.0	False	bar	2.0
c	3.0	3.0	True	bar	NaN
d	NaN	NaN	False	bar	NaN

```
In [87]: iris = pd.read_csv("Iris.csv")
iris.head()
```

```
Out[87]:   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species
```

0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

```
In [88]: iris.assign(sepal_ratio=iris["SepalWidthCm"] / iris["SepalLengthCm"]).head()
```

```
Out[88]:   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species  sepal_ratio
```

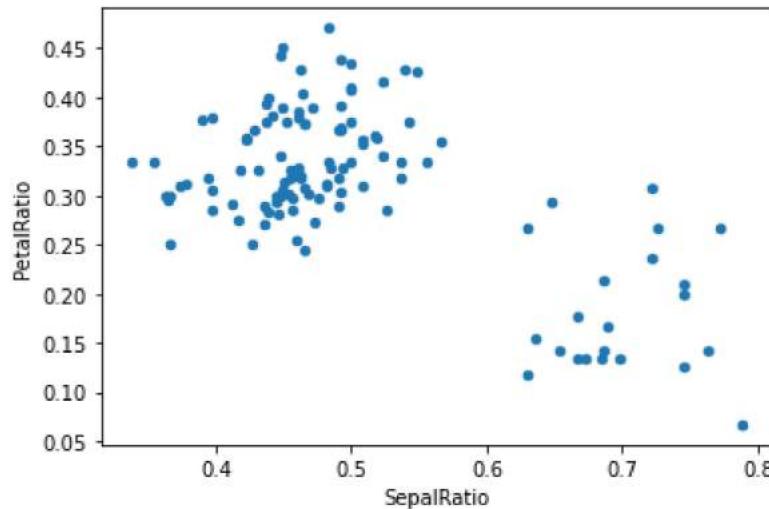
0	1	5.1	3.5	1.4	0.2	Iris-setosa	0.686275
1	2	4.9	3.0	1.4	0.2	Iris-setosa	0.612245
2	3	4.7	3.2	1.3	0.2	Iris-setosa	0.680851
3	4	4.6	3.1	1.5	0.2	Iris-setosa	0.673913
4	5	5.0	3.6	1.4	0.2	Iris-setosa	0.720000

```
In [89]: iris.assign(sepal_ratio=lambda x: (x["SepalWidthCm"] / x["SepalLengthCm"])).head()
```

```
Out[89]:   Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  Species  sepal_ratio
```

0	1	5.1	3.5	1.4	0.2	Iris-setosa	0.686275
1	2	4.9	3.0	1.4	0.2	Iris-setosa	0.612245
2	3	4.7	3.2	1.3	0.2	Iris-setosa	0.680851
3	4	4.6	3.1	1.5	0.2	Iris-setosa	0.673913
4	5	5.0	3.6	1.4	0.2	Iris-setosa	0.720000

```
In [90]: (iris.query("SepalLengthCm > 5").assign(
SepalRatio=lambda x: x.SepalWidthCm / x.SepalLengthCm,
PetalRatio=lambda x: x.PetalWidthCm / x.PetalLengthCm,
).plot(kind="scatter", x="SepalRatio", y="PetalRatio"));
```



```
In [91]: dfa = pd.DataFrame({"A": [1, 2, 3], "B": [4, 5, 6]})
dfa.assign(C=lambda x: x["A"] + x["B"], D=lambda x: x["A"] + x["C"])
```

```
Out[91]:
```

	A	B	C	D
0	1	4	5	6
1	2	5	7	9
2	3	6	9	12

```
In [92]: df = pd.DataFrame(np.random.randn(10, 4), columns=["A", "B", "C", "D"])
df2 = pd.DataFrame(np.random.randn(7, 3), columns=["A", "B", "C"])
df + df2
```

```
Out[92]:
```

	A	B	C	D
0	2.420866	1.658919	-0.004957	NaN
1	0.230904	-1.496995	0.531114	NaN
2	0.387801	-2.293522	-0.940368	NaN
3	0.198035	-1.598423	0.608666	NaN
4	-0.750591	1.431920	-1.540342	NaN
5	0.144717	-2.595985	0.175171	NaN
6	0.396199	0.681542	3.810757	NaN
7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN
9	NaN	NaN	NaN	NaN

```
In [93]: df1 = pd.DataFrame({"a": [1, 0, 1], "b": [0, 1, 1]}, dtype=bool)
df2 = pd.DataFrame({"a": [0, 1, 1], "b": [1, 1, 0]}, dtype=bool)
df1 & df2
```

```
Out[93]:      a      b
              0  False  False
              1  False   True
              2   True  False
```

```
In [94]: df1 | df2
```

```
Out[94]:      a      b
              0   True   True
              1   True   True
              2   True   True
```

```
In [95]: df1 ^ df2
```

```
Out[95]:      a      b
              0   True   True
              1   True  False
              2  False   True
```

```
In [96]: ~df1
```

```
Out[96]:      a      b
              0  False   True
              1   True  False
              2  False  False
```

```
In [97]: np.exp(df)
```

```
Out[97]:
```

	A	B	C	D
0	4.850797	7.643395	1.172958	1.574652
1	4.009842	0.933421	7.619213	0.661011
2	1.349996	0.193248	0.811424	1.976038
3	0.941819	0.520655	0.461768	2.772299
4	0.894556	1.922714	0.224244	3.327236
5	1.056101	0.233735	0.556385	0.303122
6	2.077379	0.660008	7.266333	3.149107
7	0.800964	1.414727	0.712628	0.400347
8	1.998324	2.184248	3.783204	0.802994
9	0.723489	8.619131	0.100072	2.146063

```
In [98]: ser = pd.Series([1, 2, 3, 4])
np.exp(ser)
```

```
Out[98]: 0    2.718282
1    7.389056
2    20.085537
3    54.598150
dtype: float64
```

```
In [99]: ser1 = pd.Series([1, 2, 3], index=["a", "b", "c"])
ser2 = pd.Series([1, 3, 5], index=["b", "a", "c"])
ser1
```

```
Out[99]: a    1
b    2
c    3
dtype: int64
```

```
In [100]: np.remainder(ser1, ser2)
```

```
Out[100]: a    1
b    0
c    3
dtype: int64
```

```
In [101]: ser3 = pd.Series([2, 4, 6], index=["b", "c", "d"])
np.remainder(ser1, ser3)
```

```
Out[101]: a    NaN
b    0.0
c    3.0
d    NaN
dtype: float64
```

```
In [102]: ser = pd.Series([1, 2, 3])
idx = pd.Index([4, 5, 6])
np.maximum(ser, idx)
```

```
Out[102]: 0    4
           1    5
           2    6
          dtype: int64
```

```
In [104]: pd.DataFrame(np.random.randn(3, 12))
```

```
Out[104]:   0      1      2      3      4      5      6      7      8
0 -0.501245  0.571128 -0.508366 -0.326784  0.636010  1.006448 -0.910515 -0.502634  1.306846
1 -0.555579  0.528680  0.244514 -0.921669 -0.942707 -1.253826 -1.811112 -0.900820  0.616784
2 -0.576731 -0.872153  2.207928 -0.505467 -0.130966  1.685892 -1.459214 -0.262680 -0.229437
```

```
◀ ▶
```

```
In [105]: pd.set_option("display.width", 40) # default is 80
pd.DataFrame(np.random.randn(3, 12))
```

```
Out[105]:   0      1      2      3      4      5      6      7      8
0  0.022773 -1.401868 -0.970989  2.171251  0.414099  0.151816 -1.110283 -0.449167  1.714537
1 -0.789318 -0.265809 -0.678194 -2.483981  0.364979 -0.704912 -0.847870 -1.562679  1.817926
2 -0.841586 -1.769016  1.341991 -0.463400  1.139975  0.237341 -0.223064 -0.146064  0.253593
```

```
◀ ▶
```

```
In [106]: datafile = {
    "filename": ["filename_01", "filename_02"],
    "path": [
        "media/user_name/storage/folder_01/filename_01",
        "media/user_name/storage/folder_02/filename_02",
    ]
}

pd.set_option("display.max_colwidth", 30)
pd.DataFrame(datafile)
```

```
Out[106]:   filename          path
0  filename_01  media/user_name/storage/folder_01/filename_01
1  filename_02  media/user_name/storage/folder_02/filename_02
```

```
In [107]: pd.set_option("display.max_colwidth", 100)
pd.DataFrame(datafile)
```

```
Out[107]:   filename          path
0  filename_01  media/user_name/storage/folder_01/filename_01
1  filename_02  media/user_name/storage/folder_02/filename_02
```

```
In [ ]: df = pd.DataFrame({"foo1": np.random.randn(5), "foo2": np.random.randn(5)})
```

```
In [108]: index = pd.date_range("1/1/2000", periods=8)
s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
df = pd.DataFrame(np.random.randn(8, 3), index=index, columns=["A", "B", "C"])
```

```
In [109]: long_series = pd.Series(np.random.randn(1000))
long_series.head()
```

```
Out[109]: 0    -0.026014
1    -1.002232
2     0.435552
3    -0.516468
4     0.919732
dtype: float64
```

```
In [110]: df[:2]
```

```
Out[110]:
```

	A	B	C
2000-01-01	-0.896867	0.519293	0.574513
2000-01-02	1.499996	0.210594	0.004202

```
In [111]: df.columns = [x.lower() for x in df.columns]
df
```

```
Out[111]:
```

	a	b	c
2000-01-01	-0.896867	0.519293	0.574513
2000-01-02	1.499996	0.210594	0.004202
2000-01-03	0.670616	0.012021	-1.118078
2000-01-04	-0.708142	-0.351169	-0.596160
2000-01-05	0.571710	-1.264462	-0.999771
2000-01-06	-0.355902	-0.458909	1.478698
2000-01-07	0.242235	0.194339	-0.864089
2000-01-08	0.073826	0.314112	1.816110

```
In [112]: s.array
```

```
Out[112]: <PandasArray>
[-0.9915161162074533,
 0.816408648335188,
 1.1267915421666856,
 0.48624698933925486,
 0.26060288152211175]
Length: 5, dtype: float64
```

```
In [113]: s.index.array
```

```
Out[113]: <PandasArray>
['a', 'b', 'c', 'd', 'e']
Length: 5, dtype: object
```

```
In [114]: s.to_numpy()
```

```
Out[114]: array([-0.99151612,  0.81640865,  1.12679154,  0.48624699,  0.26060288])
```

```
In [115]: np.asarray(s)
```

```
Out[115]: array([-0.99151612,  0.81640865,  1.12679154,  0.48624699,  0.26060288])
```

```
In [116]: ser = pd.Series(pd.date_range("2000", periods=2, tz="CET"))
ser.to_numpy(dtype=object)
```

```
Out[116]: array([Timestamp('2000-01-01 00:00:00+0100', tz='CET'),
                  Timestamp('2000-01-02 00:00:00+0100', tz='CET')], dtype=object)
```

```
In [117]: pd.set_option("compute.use_bottleneck", False)
pd.set_option("compute.use_numexpr", False)
```

```
In [118]: df = pd.DataFrame({
    "one": pd.Series(np.random.randn(3), index=["a", "b", "c"]),
    "two": pd.Series(np.random.randn(4), index=["a", "b", "c", "d"]),
    "three": pd.Series(np.random.randn(3), index=["b", "c", "d"]),
})
df
```

```
Out[118]:      one      two      three
a -1.171896 -1.181811      NaN
b  0.758395 -0.897135 -1.107687
c -0.844188  0.018352  2.354688
d      NaN    1.613328 -0.269916
```

```
In [119]: row = df.iloc[1]
column = df["two"]
df.sub(row, axis="columns")
```

```
Out[119]:      one      two      three
a -1.930291 -0.284676      NaN
b  0.000000  0.000000  0.000000
c -1.602583  0.915488  3.462375
d      NaN    2.510464  0.837772
```

```
In [120]: df.sub(row, axis=1)
```

```
Out[120]:
```

	one	two	three
a	-1.930291	-0.284676	NaN
b	0.000000	0.000000	0.000000
c	-1.602583	0.915488	3.462375
d	NaN	2.510464	0.837772

```
In [121]: df.sub(column, axis="index")
```

```
Out[121]:
```

	one	two	three
a	0.009915	0.0	NaN
b	1.655531	0.0	-0.210552
c	-0.862540	0.0	2.336335
d	NaN	0.0	-1.883244

```
In [122]: df.sub(column, axis=0)
```

```
Out[122]:
```

	one	two	three
a	0.009915	0.0	NaN
b	1.655531	0.0	-0.210552
c	-0.862540	0.0	2.336335
d	NaN	0.0	-1.883244

```
In [123]: dfmi = df.copy()
```

```
In [27]: dfmi.index = pd.MultiIndex.from_tuples(  
[(1, "a"), (1, "b"), (1, "c"), (2, "a")], names=["first", "second"]  
)
```

```
dfmi.sub(column, axis=0, level="second")
```

```
Out[123]:
```

	first	second	one	two	three
			0.009915	0.000000	NaN
1		b	1.655531	0.000000	-0.210552
		c	-0.862540	0.000000	2.336335
2		a	NaN	2.795139	0.911896

```
In [124]: pd.Series(np.arange(10))
```

```
Out[124]: 0    0  
1    1  
2    2  
3    3  
4    4  
5    5  
6    6  
7    7  
8    8  
9    9  
dtype: int32
```

```
In [125]: div, rem = divmod(df, 3)  
div
```

```
Out[125]:   one  two  three  
_____  
a   -1.0 -1.0   NaN  
b    0.0 -1.0  -1.0  
c   -1.0  0.0   0.0  
d    NaN  0.0  -1.0
```

```
In [126]: idx = pd.Index(np.arange(10))  
idx
```

```
Out[126]: Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8,  
9],  
dtype='int64')
```

```
In [127]: div, rem = divmod(idx, 3)  
div
```

```
Out[127]: Int64Index([0, 0, 0, 1, 1, 1, 2, 2, 2,  
3],  
dtype='int64')
```

```
In [128]: df.gt(df)
```

```
Out[128]:   one  two  three  
_____  
a  False False False  
b  False False False  
c  False False False  
d  False False False
```

```
In [129]: (df > 0).all()
```

```
Out[129]: one      False
           two      False
           three     False
          dtype: bool
```

```
In [130]: (df > 0).any()
```

```
Out[130]: one      True
           two      True
           three     True
          dtype: bool
```

```
In [131]: (df > 0).any().any()
```

```
Out[131]: True
```

```
In [132]: pd.DataFrame(columns=list("ABC")).empty
```

```
Out[132]: True
```

```
In [133]: (df + df == df * 2).all()
```

```
Out[133]: one      False
           two      True
           three     False
          dtype: bool
```

```
In [134]: (df + df).equals(df * 2)
```

```
Out[134]: True
```

```
In [135]: df1 = pd.DataFrame({"col": ["foo", 0, np.nan]})  
df2 = pd.DataFrame({"col": [np.nan, 0, "foo"]}, index=[2, 1, 0])  
df1.equals(df2)
```

```
Out[135]: False
```

```
In [136]: pd.Series(["foo", "bar", "baz"]) == "foo"
```

```
Out[136]: 0      True
           1      False
           2      False
          dtype: bool
```

```
In [137]: pd.Index(["foo", "bar", "baz"]) == "foo"
```

```
Out[137]: array([ True, False, False])
```

```
In [138]: pd.Series(["foo", "bar", "baz"]) == pd.Index(["foo", "bar", "qux"])
```

```
Out[138]: 0    True
1    True
2   False
dtype: bool
```

```
In [139]: pd.Series(["foo", "bar", "baz"]) == np.array(["foo", "bar", "qux"])
```

```
Out[139]: 0    True
1    True
2   False
dtype: bool
```

```
In [140]: np.array([1, 2, 3]) == np.array([2])
```

```
Out[140]: array([False,  True, False])
```

```
In [141]: df1 = pd.DataFrame(
    {"A": [1.0, np.nan, 3.0, 5.0, np.nan], "B": [np.nan, 2.0, 3.0, np.nan, 6.0]}
)

df2 = pd.DataFrame({
    "A": [5.0, 2.0, 4.0, np.nan, 3.0, 7.0],
    "B": [np.nan, np.nan, 3.0, 4.0, 6.0, 8.0],
})

df1
```

```
Out[141]:      A      B
0    1.0    NaN
1    NaN    2.0
2    3.0    3.0
3    5.0    NaN
4    NaN    6.0
```

```
In [142]: df1.combine_first(df2)
```

```
Out[142]:      A      B
0    1.0    NaN
1    2.0    2.0
2    3.0    3.0
3    5.0    4.0
4    3.0    6.0
5    7.0    8.0
```

```
In [143]: def combiner(x, y):
    return np.where(pd.isna(x), y, x)
df1.combine(df2, combiner)
```

```
Out[143]:
```

	A	B
0	1.0	NaN
1	2.0	2.0
2	3.0	3.0
3	5.0	4.0
4	3.0	6.0
5	7.0	8.0

```
In [144]: df.sum(0, skipna=False)
```

```
Out[144]: one      NaN
two     -0.447266
three      NaN
dtype: float64
```

```
In [145]: df.sum(axis=1, skipna=True)
```

```
Out[145]: a     -2.353707
b     -1.246427
c      1.528852
d      1.343413
dtype: float64
```

```
In [146]: ts_stand = (df - df.mean()) / df.std()
ts_stand.std()
```

```
Out[146]: one      1.0
two      1.0
three     1.0
dtype: float64
```

```
In [147]: xs_stand = df.sub(df.mean(1), axis=0).div(df.std(1), axis=0)
xs_stand.std(1)
```

```
Out[147]: a      1.0
b      1.0
c      1.0
d      1.0
dtype: float64
```

```
In [148]: np.mean(df["one"])
```

```
Out[148]: -0.41922927314676367
```

```
In [149]: np.mean(df["one"].to_numpy())
```

```
Out[149]: nan
```

```
In [150]: series = pd.Series(np.random.randn(500))
series[20:500] = np.nan
series[10:20] = 5
series.unique()
```

```
Out[150]: 11
```

```
In [151]: series = pd.Series(np.random.randn(1000))
series[::2] = np.nan
series.describe()
```

```
Out[151]: count    500.000000
mean     -0.007995
std      0.963051
min     -4.036894
25%     -0.617967
50%      0.044174
75%      0.671260
max      2.694971
dtype: float64
```

```
In [152]: frame = pd.DataFrame(np.random.randn(1000, 5), columns=["a", "b", "c", "d", "e"]
frame.iloc[::2] = np.nan
frame.describe()
```

```
Out[152]:
```

	a	b	c	d	e
count	500.000000	500.000000	500.000000	500.000000	500.000000
mean	-0.033536	-0.000459	0.063157	0.059826	0.043546
std	0.973013	0.936975	1.046047	1.024661	0.978513
min	-3.491812	-2.591516	-2.787851	-2.796227	-3.234868
25%	-0.637533	-0.607886	-0.712782	-0.616478	-0.564365
50%	-0.022158	0.024663	0.037883	0.044762	-0.024644
75%	0.666086	0.639350	0.758282	0.721884	0.685598
max	2.566799	3.027822	2.766063	3.472135	3.077083

```
In [153]: series.describe(percentiles=[0.05, 0.25, 0.75, 0.95])
```

```
Out[153]: count    500.000000
mean     -0.007995
std      0.963051
min     -4.036894
5%      -1.707012
25%     -0.617967
50%     0.044174
75%     0.671260
95%     1.444216
max      2.694971
dtype: float64
```

```
In [154]: s = pd.Series(["a", "a", "b", "b", "a", "a", np.nan, "c", "d", "a"])
s.describe()
```

```
Out[154]: count    9
unique    4
top      a
freq      5
dtype: object
```

```
In [1]: import pandas as pd  
import numpy as np
```

```
In [2]: frame = pd.DataFrame({"a": ["Yes", "Yes", "No", "No"], "b": range(4)})  
frame.describe()
```

Out[2]:

	b
count	4.000000
mean	1.500000
std	1.290994
min	0.000000
25%	0.750000
50%	1.500000
75%	2.250000
max	3.000000

```
In [3]: frame.describe(include=["object"])
```

Out[3]:

	a
count	4
unique	2
top	Yes
freq	2

```
In [6]: frame.describe(include=["number"])
```

```
Out[6]:      b  
_____  
count    4.000000  
mean     1.500000  
std      1.290994  
min     0.000000  
25%    0.750000  
50%    1.500000  
75%    2.250000  
max     3.000000
```

```
In [7]: s1 = pd.Series(np.random.randn(5))  
s1
```

```
Out[7]: 0   -0.121086  
1    0.060713  
2    1.259896  
3   -0.161383  
4   -2.168469  
dtype: float64
```

```
In [8]: s1.idxmin(), s1.idxmax()
```

```
Out[8]: (4, 2)
```

```
In [9]: df1 = pd.DataFrame(np.random.randn(5, 3), columns=["A", "B", "C"])  
df1
```

```
Out[9]:      A         B         C  
_____  
0  -1.029309  1.362440 -0.959433  
1   0.862846 -0.221771 -1.559672  
2   0.735617  0.847179 -0.020883  
3  -1.213478  0.416975  1.226910  
4   0.020545 -0.211762 -1.391545
```

```
In [10]: df1.idxmin(axis=0)
```

```
Out[10]: A    3  
B    1  
C    1  
dtype: int64
```

```
In [11]: df1.idxmax(axis=1)
```

```
Out[11]: 0    B  
1    A  
2    B  
3    C  
4    A  
dtype: object
```

```
In [12]: df3 = pd.DataFrame([2, 1, 1, 3, np.nan], columns=["A"], index=list("edcba"))  
df3
```

```
Out[12]:      A  
_____  
e    2.0  
d    1.0  
c    1.0  
b    3.0  
a    NaN
```

```
In [13]: data = np.random.randint(0, 7, size=50)  
data
```

```
Out[13]: array([6, 5, 6, 2, 6, 2, 3, 5, 1, 3, 1, 3, 1, 5, 2, 6, 6, 4, 4, 6, 0, 5,  
0, 3, 5, 4, 1, 2, 2, 6, 1, 6, 1, 0, 4, 4, 0, 4, 3, 5, 6, 0, 6, 4,  
5, 5, 1, 1, 2, 5])
```

```
In [14]: s = pd.Series(data)  
s.value_counts()
```

```
Out[14]: 6    10  
5     9  
1     8  
4     7  
2     6  
3     5  
0     5  
dtype: int64
```

```
In [15]: data = {"a": [1, 2, 3, 4], "b": ["x", "x", "y", "y"]}  
frame = pd.DataFrame(data)  
frame.value_counts()
```

```
Out[15]:   a   b  
1   x    1  
2   x    1  
3   y    1  
4   y    1  
dtype: int64
```

```
In [16]: s5 = pd.Series([1, 1, 3, 3, 3, 5, 5, 7, 7, 7])
s5.mode()
```

```
Out[16]: 0    3
          1    7
          dtype: int64
```

```
In [17]: df5 = pd.DataFrame({
    "A": np.random.randint(0, 7, size=50),
    "B": np.random.randint(-10, 15, size=50),
})
df5.mode()
```

```
Out[17]:      A   B
0   0.0 -9
1   NaN -3
```

```
In [18]: arr = np.random.randn(20)
factor = pd.cut(arr, 4)
factor
```

```
Out[18]: [(-0.245, 0.809], (0.809, 1.863], (-1.303, -0.245], (-0.245, 0.809], (-0.245, 0.809], ...,
           (-0.245, 0.809], (-0.245, 0.809], (1.863, 2.917], (-0.245, 0.809], (-0.245, 0.809])
Length: 20
Categories (4, interval[float64, right]): [(-1.303, -0.245] < (-0.245, 0.809]
< (0.809, 1.863] < (1.863, 2.917]]
```

```
In [19]: factor = pd.cut(arr, [-5, -1, 0, 1, 5])
factor
```

```
Out[19]: [(0, 1], (1, 5], (-1, 0], (-1, 0], (0, 1], ..., (0, 1], (0, 1], (1, 5], (-1, 0],
           (-1, 0])
Length: 20
Categories (4, interval[int64, right]): [(-5, -1] < (-1, 0] < (0, 1] < (1, 5]]
```

```
In [20]: arr = np.random.randn(30)
factor = pd.qcut(arr, [0, 0.25, 0.5, 0.75, 1])
factor
```

```
Out[20]: [(-0.204, 0.428], (-0.75, -0.204], (-3.0669999999999997, -0.75], (-0.75, -0.204],
           (-0.75, -0.204], ..., (-3.0669999999999997, -0.75], (-0.204, 0.428], (-3.0669999999999997,
           -0.75], (-0.204, 0.428], (-0.75, -0.204])
Length: 30
Categories (4, interval[float64, right]): [(-3.0669999999999997, -0.75] < (-0.75, -0.204]
< (-0.204, 0.428] < (0.428, 2.156]]
```

```
In [21]: arr = np.random.randn(20)
factor = pd.cut(arr, [-np.inf, 0, np.inf])
factor
```

```
Out[21]: [(-inf, 0.0], (-inf, 0.0], (0.0, inf], (0.0, inf], (0.0, inf], ..., (-inf, 0.0], (0.0, inf], (0.0, inf], (-inf, 0.0], (-inf, 0.0])
Length: 20
Categories (2, interval[float64, right]): [(-inf, 0.0] < (0.0, inf]]
```

```
In [23]: def extract_city_name(df):
    df["city_name"] = df["city_and_code"].str.split(",").str.get(0)
    return df
def add_country_name(df, country_name=None):
    col = "city_name"
    df["city_and_country"] = df[col] + country_name
    return df
df_p = pd.DataFrame({"city_and_code": ["Chicago, IL"]})

add_country_name(extract_city_name(df_p), country_name="US")
```

```
Out[23]:   city_and_code  city_name  city_and_country
          0      Chicago, IL      Chicago      ChicagoUS
```

```
In [24]: df_p.pipe(extract_city_name).pipe(add_country_name, country_name="US")
```

```
Out[24]:   city_and_code  city_name  city_and_country
          0      Chicago, IL      Chicago      ChicagoUS
```

```
In [25]: import statsmodels.formula.api as sm
bb = pd.read_csv("baseball.csv", index_col="id")
(
bb.query("h > 0")
.assign(ln_h=lambda df: np.log(df.h))
.pipe((sm.ols, "data"), "hr ~ ln_h + year + g + C(lg)")
.fit()
.summary()
)
```

Out[25]: OLS Regression Results

Dep. Variable:	hr	R-squared:	0.685			
Model:	OLS	Adj. R-squared:	0.665			
Method:	Least Squares	F-statistic:	34.28			
Date:	Thu, 22 Sep 2022	Prob (F-statistic):	3.48e-15			
Time:	18:53:34	Log-Likelihood:	-205.92			
No. Observations:	68	AIC:	421.8			
Df Residuals:	63	BIC:	432.9			
Df Model:	4					
Covariance Type:	nonrobust					
	coef	std err	t	P> t	[0.025	0.975]
Intercept	-8484.7720	4664.146	-1.819	0.074	-1.78e+04	835.780
C(lg)[T.NL]	-2.2736	1.325	-1.716	0.091	-4.922	0.375
ln_h	-1.3542	0.875	-1.547	0.127	-3.103	0.395
year	4.2277	2.324	1.819	0.074	-0.417	8.872
g	0.1841	0.029	6.258	0.000	0.125	0.243
Omnibus:	10.875	Durbin-Watson:	1.999			
Prob(Omnibus):	0.004	Jarque-Bera (JB):	17.298			
Skew:	0.537	Prob(JB):	0.000175			
Kurtosis:	5.225	Cond. No.	1.49e+07			

Notes:

- [1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
- [2] The condition number is large, 1.49e+07. This might indicate that there are strong multicollinearity or other numerical problems.

In [27]: df5.apply(np.mean)

Out[27]: A 2.96
B 2.06
dtype: float64

```
In [28]: df5.apply(np.mean, axis=1)
```

```
Out[28]: 0      3.0
1      4.5
2      6.0
3      1.0
4      3.0
5      6.5
6     -2.5
7     -4.5
8      5.0
9      8.5
10     2.0
11     0.0
12     3.0
13     2.0
14    -1.5
15     5.5
16     6.5
17    -3.5
18     2.5
19     0.5
20     1.0
21     6.5
22    -1.5
23     3.0
24    -3.5
25     7.5
26     5.0
27    -2.5
28     2.0
29     3.0
30     8.5
31     4.5
32     0.0
33     9.0
34     6.5
35     8.0
36    -2.5
37     0.0
38     1.5
39     0.5
40    -4.0
41     0.5
42     7.5
43    -0.5
44     0.0
45     7.0
46     4.0
47     5.0
48     1.0
49     1.0
dtype: float64
```

```
In [29]: df5.apply(lambda x: x.max() - x.min())
```

```
Out[29]: A      6  
B     23  
dtype: int64
```

In [30]: `df5.apply(np.cumsum)`

Out[30]:

	A	B
0	0	6
1	2	13
2	2	25
3	7	22
4	7	28
5	12	36
6	15	28
7	15	19
8	17	27
9	21	40
10	22	43
11	22	43
12	25	46
13	31	44
14	37	35
15	39	44
16	41	55
17	44	45
18	50	44
19	51	44
20	51	46
21	56	54
22	61	46
23	62	51
24	64	42
25	70	51
26	73	58
27	78	48
28	83	47
29	87	49
30	93	60
31	93	69
32	97	65
33	103	77
34	104	89
35	110	99

	A	B
36	111	93
37	114	90
38	119	88
39	123	85
40	124	76
41	128	73
42	132	84
43	138	77
44	143	72
45	144	85
46	146	91
47	146	101
48	146	103
49	148	103

In [32]: `df5.apply(np.exp)`

Out[32]:

	A	B
0	1.000000	403.428793
1	7.389056	1096.633158
2	1.000000	162754.791419
3	148.413159	0.049787
4	1.000000	403.428793
5	148.413159	2980.957987
6	20.085537	0.000335
7	1.000000	0.000123
8	7.389056	2980.957987
9	54.598150	442413.392009
10	2.718282	20.085537
11	1.000000	1.000000
12	20.085537	20.085537
13	403.428793	0.135335
14	403.428793	0.000123
15	7.389056	8103.083928
16	7.389056	59874.141715
17	20.085537	0.000045
18	403.428793	0.367879
19	2.718282	1.000000
20	1.000000	7.389056
21	148.413159	2980.957987
22	148.413159	0.000335
23	2.718282	148.413159
24	7.389056	0.000123
25	403.428793	8103.083928
26	20.085537	1096.633158
27	148.413159	0.000045
28	148.413159	0.367879
29	54.598150	7.389056
30	403.428793	59874.141715
31	1.000000	8103.083928
32	54.598150	0.018316
33	403.428793	162754.791419
34	2.718282	162754.791419
35	403.428793	22026.465795

	A	B
36	2.718282	0.002479
37	20.085537	0.049787
38	148.413159	0.135335
39	54.598150	0.049787
40	2.718282	0.000123
41	54.598150	0.049787
42	54.598150	59874.141715
43	403.428793	0.000912
44	148.413159	0.006738
45	2.718282	442413.392009
46	7.389056	403.428793
47	1.000000	22026.465795
48	1.000000	7.389056
49	7.389056	1.000000

In [33]: `df5.apply("mean")`

Out[33]: A 2.96
B 2.06
dtype: float64

```
In [35]: df5.apply("mean", axis=1)
```

```
Out[35]: 0      3.0
1      4.5
2      6.0
3      1.0
4      3.0
5      6.5
6     -2.5
7     -4.5
8      5.0
9      8.5
10     2.0
11     0.0
12     3.0
13     2.0
14    -1.5
15     5.5
16     6.5
17    -3.5
18     2.5
19     0.5
20     1.0
21     6.5
22    -1.5
23     3.0
24    -3.5
25     7.5
26     5.0
27    -2.5
28     2.0
29     3.0
30     8.5
31     4.5
32     0.0
33     9.0
34     6.5
35     8.0
36    -2.5
37     0.0
38     1.5
39     0.5
40    -4.0
41     0.5
42     7.5
43    -0.5
44     0.0
45     7.0
46     4.0
47     5.0
48     1.0
49     1.0
dtype: float64
```

```
In [36]: def subtract_and_divide(x, sub, divide=1):
    return (x - sub) / divide

df5.apply(subtract_and_divide, args=(5,), divide=3)
```

Out[36]:

	A	B
0	-1.666667	0.333333
1	-1.000000	0.666667
2	-1.666667	2.333333
3	0.000000	-2.666667
4	-1.666667	0.333333
5	0.000000	1.000000
6	-0.666667	-4.333333
7	-1.666667	-4.666667
8	-1.000000	1.000000
9	-0.333333	2.666667
10	-1.333333	-0.666667
11	-1.666667	-1.666667
12	-0.666667	-0.666667
13	0.333333	-2.333333
14	0.333333	-4.666667
15	-1.000000	1.333333
16	-1.000000	2.000000
17	-0.666667	-5.000000
18	0.333333	-2.000000
19	-1.333333	-1.666667
20	-1.666667	-1.000000
21	0.000000	1.000000
22	0.000000	-4.333333
23	-1.333333	0.000000
24	-1.000000	-4.666667
25	0.333333	1.333333
26	-0.666667	0.666667
27	0.000000	-5.000000
28	0.000000	-2.000000
29	-0.333333	-1.000000
30	0.333333	2.000000
31	-1.666667	1.333333
32	-0.333333	-3.000000
33	0.333333	2.333333
34	-1.333333	2.333333
35	0.333333	1.666667

	A	B
36	-1.333333	-3.666667
37	-0.666667	-2.666667
38	0.000000	-2.333333
39	-0.333333	-2.666667
40	-1.333333	-4.666667
41	-0.333333	-2.666667
42	-0.333333	2.000000
43	0.333333	-4.000000
44	0.000000	-3.333333
45	-1.333333	2.666667
46	-1.000000	0.333333
47	-1.666667	1.666667
48	-1.666667	-1.000000
49	-1.000000	-1.666667

```
In [37]: tsdf = pd.DataFrame(
    np.random.randn(10, 3),
    columns=["A", "B", "C"],
    index=pd.date_range("1/1/2000", periods=10),
)

tsdf.iloc[3:7] = np.nan
tsdf
```

	A	B	C
2000-01-01	1.100146	-0.594632	-0.486077
2000-01-02	-1.281338	-0.032859	0.675010
2000-01-03	-1.250284	1.207627	-0.363746
2000-01-04	NaN	NaN	NaN
2000-01-05	NaN	NaN	NaN
2000-01-06	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN
2000-01-08	1.381345	-1.236094	2.241808
2000-01-09	0.209783	0.166198	-0.248163
2000-01-10	2.156381	0.918886	-2.077679

```
In [38]: tsdf.agg(np.sum)
```

```
Out[38]: A    2.316033
          B    0.429125
          C   -0.258846
          dtype: float64
```

```
In [39]: tsdf.agg("sum")
```

```
Out[39]: A    2.316033
          B    0.429125
          C   -0.258846
          dtype: float64
```

```
In [40]: tsdf.sum()
```

```
Out[40]: A    2.316033
          B    0.429125
          C   -0.258846
          dtype: float64
```

```
In [41]: tsdf["A"].agg("sum")
```

```
Out[41]: 2.3160328735804745
```

```
In [42]: tsdf.agg(["sum", "mean"])
```

```
Out[42]:      A      B      C
sum  2.316033  0.429125 -0.258846
mean 0.386005  0.071521 -0.043141
```

```
In [43]: tsdf["A"].agg(["sum", "mean"])
```

```
Out[43]: sum      2.316033
          mean     0.386005
          Name: A, dtype: float64
```

```
In [44]: tsdf["A"].agg(["sum", lambda x: x.mean()])
```

```
Out[44]: sum      2.316033
          <lambda>  0.386005
          Name: A, dtype: float64
```

```
In [45]: def mymean(x):
          return x.mean()

tsdf["A"].agg(["sum", mymean])
```

```
Out[45]: sum      2.316033
          mymean  0.386005
          Name: A, dtype: float64
```

```
In [46]: tsdf.agg({"A": "mean", "B": "sum"})
```

```
Out[46]: A    0.386005  
B    0.429125  
dtype: float64
```

```
In [47]: tsdf.agg({"A": ["mean", "min"], "B": "sum"})
```

```
Out[47]:      A      B  
mean  0.386005    NaN  
min  -1.281338    NaN  
sum       NaN  0.429125
```

```
In [48]: mdf = pd.DataFrame({  
    "A": [1, 2, 3],  
    "B": [1.0, 2.0, 3.0],  
    "C": ["foo", "bar", "baz"],  
    "D": pd.date_range("20130101", periods=3),  
})
```

```
In [49]: mdf.agg(["min", "sum"])
```

```
Out[49]:      A      B      C      D  
min   1  1.0    bar 2013-01-01  
sum   6  6.0  foobarbaz        NaT
```

```
In [50]: # Custom describe
```

```
from functools import partial  
q_25 = partial(pd.Series.quantile, q=0.25)  
q_25.__name__ = "25%"  
q_75 = partial(pd.Series.quantile, q=0.75)  
q_75.__name__ = "75%"  
tsdf.agg(["count", "mean", "std", "min", q_25, "median", q_75, "max"])
```

```
Out[50]:      A      B      C  
count  6.000000  6.000000  6.000000  
mean   0.386005  0.071521 -0.043141  
std    1.422916  0.914575  1.429482  
min   -1.281338 -1.236094 -2.077679  
25%   -0.885267 -0.454189 -0.455494  
median  0.654965  0.066669 -0.305954  
75%   1.311045  0.730714  0.444217  
max   2.156381  1.207627  2.241808
```

```
In [52]: tsdf = pd.DataFrame(  
    np.random.randn(10, 3),  
    columns=["A", "B", "C"],  
    index=pd.date_range("1/1/2000", periods=10),  
)  
  
tsdf.iloc[3:7] = np.nan  
tsdf
```

Out[52]:

	A	B	C
2000-01-01	-0.632673	0.474561	-0.798479
2000-01-02	1.250986	-0.578337	1.065323
2000-01-03	-0.998635	-1.218509	-0.738105
2000-01-04	NaN	NaN	NaN
2000-01-05	NaN	NaN	NaN
2000-01-06	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN
2000-01-08	-0.683195	1.623150	0.090563
2000-01-09	-0.118824	-0.426729	-1.098490
2000-01-10	-1.150192	0.214560	1.337532

```
In [53]: tsdf.transform(np.abs)
```

Out[53]:

	A	B	C
2000-01-01	0.632673	0.474561	0.798479
2000-01-02	1.250986	0.578337	1.065323
2000-01-03	0.998635	1.218509	0.738105
2000-01-04	NaN	NaN	NaN
2000-01-05	NaN	NaN	NaN
2000-01-06	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN
2000-01-08	0.683195	1.623150	0.090563
2000-01-09	0.118824	0.426729	1.098490
2000-01-10	1.150192	0.214560	1.337532

```
In [54]: tsdf.transform("abs")
```

Out[54]:

	A	B	C
2000-01-01	0.632673	0.474561	0.798479
2000-01-02	1.250986	0.578337	1.065323
2000-01-03	0.998635	1.218509	0.738105
2000-01-04	NaN	NaN	NaN
2000-01-05	NaN	NaN	NaN
2000-01-06	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN
2000-01-08	0.683195	1.623150	0.090563
2000-01-09	0.118824	0.426729	1.098490
2000-01-10	1.150192	0.214560	1.337532

```
In [55]: tsdf.transform(lambda x: x.abs())
```

Out[55]:

	A	B	C
2000-01-01	0.632673	0.474561	0.798479
2000-01-02	1.250986	0.578337	1.065323
2000-01-03	0.998635	1.218509	0.738105
2000-01-04	NaN	NaN	NaN
2000-01-05	NaN	NaN	NaN
2000-01-06	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN
2000-01-08	0.683195	1.623150	0.090563
2000-01-09	0.118824	0.426729	1.098490
2000-01-10	1.150192	0.214560	1.337532

```
In [56]: np.abs(tsdf)
```

```
Out[56]:
```

	A	B	C
2000-01-01	0.632673	0.474561	0.798479
2000-01-02	1.250986	0.578337	1.065323
2000-01-03	0.998635	1.218509	0.738105
2000-01-04	NaN	NaN	NaN
2000-01-05	NaN	NaN	NaN
2000-01-06	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN
2000-01-08	0.683195	1.623150	0.090563
2000-01-09	0.118824	0.426729	1.098490
2000-01-10	1.150192	0.214560	1.337532

```
In [57]: tsdf["A"].transform(np.abs)
```

```
Out[57]:
```

2000-01-01	0.632673
2000-01-02	1.250986
2000-01-03	0.998635
2000-01-04	NaN
2000-01-05	NaN
2000-01-06	NaN
2000-01-07	NaN
2000-01-08	0.683195
2000-01-09	0.118824
2000-01-10	1.150192

Freq: D, Name: A, dtype: float64

```
In [58]: tsdf.transform([np.abs, lambda x: x + 1])
```

```
Out[58]:
```

	A	B	C			
	absolute	<lambda>	absolute	<lambda>	absolute	<lambda>
2000-01-01	0.632673	0.367327	0.474561	1.474561	0.798479	0.201521
2000-01-02	1.250986	2.250986	0.578337	0.421663	1.065323	2.065323
2000-01-03	0.998635	0.001365	1.218509	-0.218509	0.738105	0.261895
2000-01-04	NaN	NaN	NaN	NaN	NaN	NaN
2000-01-05	NaN	NaN	NaN	NaN	NaN	NaN
2000-01-06	NaN	NaN	NaN	NaN	NaN	NaN
2000-01-07	NaN	NaN	NaN	NaN	NaN	NaN
2000-01-08	0.683195	0.316805	1.623150	2.623150	0.090563	1.090563
2000-01-09	0.118824	0.881176	0.426729	0.573271	1.098490	-0.098490
2000-01-10	1.150192	-0.150192	0.214560	1.214560	1.337532	2.337532

```
In [59]: tsdf["A"].transform([np.abs, lambda x: x + 1])
```

Out[59]:

	absolute	<lambda>
2000-01-01	0.632673	0.367327
2000-01-02	1.250986	2.250986
2000-01-03	0.998635	0.001365
2000-01-04	NaN	NaN
2000-01-05	NaN	NaN
2000-01-06	NaN	NaN
2000-01-07	NaN	NaN
2000-01-08	0.683195	0.316805
2000-01-09	0.118824	0.881176
2000-01-10	1.150192	-0.150192

```
In [60]: tsdf.transform({"A": np.abs, "B": lambda x: x + 1})
```

Out[60]:

	A	B
2000-01-01	0.632673	1.474561
2000-01-02	1.250986	0.421663
2000-01-03	0.998635	-0.218509
2000-01-04	NaN	NaN
2000-01-05	NaN	NaN
2000-01-06	NaN	NaN
2000-01-07	NaN	NaN
2000-01-08	0.683195	2.623150
2000-01-09	0.118824	0.573271
2000-01-10	1.150192	1.214560

```
In [68]: df = pd.DataFrame({
    "one": pd.Series(np.random.randn(3), index=["a", "b", "c"]),
    "two": pd.Series(np.random.randn(4), index=["a", "b", "c", "d"]),
    "three": pd.Series(np.random.randn(3), index=["b", "c", "d"]),
})
```

df

Out[68]:

	one	two	three
a	-0.789990	-0.020558	NaN
b	0.572949	-0.778513	-0.450913
c	-0.367262	1.963174	-0.203882
d	NaN	-0.313509	0.001874

```
In [69]: df["three"] = df["one"] * df["two"]
df["flag"] = df["one"] > 2
df
```

```
Out[69]:      one      two      three   flag
a -0.789990 -0.020558  0.016241  False
b  0.572949 -0.778513 -0.446048  False
c -0.367262  1.963174 -0.721000  False
d       NaN -0.313509       NaN  False
```

```
In [70]: def f(x):
          return len(str(x))
df["one"].map(f)
```

```
Out[70]: a    19
b    18
c    19
d     3
Name: one, dtype: int64
```

```
In [71]: df.applymap(f)
```

```
Out[71]:      one  two  three   flag
a    19   21    19     5
b    18   19    19     5
c    19   18    19     5
d     3   19     3     5
```

```
In [72]: s = pd.Series(
           ["six", "seven", "six", "seven", "six"], index=["a", "b", "c", "d", "e"]
         )

t = pd.Series({"six": 6.0, "seven": 7.0})
s
```

```
Out[72]: a      six
b    seven
c      six
d    seven
e      six
dtype: object
```

```
In [73]: s.map(t)
```

```
Out[73]: a    6.0
          b    7.0
          c    6.0
          d    7.0
          e    6.0
          dtype: float64
```

```
In [74]: s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
          s
```

```
Out[74]: a    -0.727970
          b     1.102445
          c     1.113834
          d    -0.195407
          e     0.238101
          dtype: float64
```

```
In [75]: s.reindex(["e", "b", "f", "d"])
```

```
Out[75]: e    0.238101
          b     1.102445
          f      NaN
          d    -0.195407
          dtype: float64
```

```
In [76]: df.reindex(index=["c", "f", "b"], columns=["three", "two", "one"])
```

```
Out[76]:      three      two      one
          c -0.721000  1.963174 -0.367262
          f      NaN       NaN       NaN
          b -0.446048 -0.778513  0.572949
```

```
In [77]: df.reindex(["c", "f", "b"], axis="index")
```

```
Out[77]:      one      two      three   flag
          c -0.367262  1.963174 -0.721000  False
          f      NaN       NaN       NaN    NaN
          b   0.572949 -0.778513 -0.446048  False
```

```
In [78]: rs = s.reindex(df.index)
          rs
```

```
Out[78]: a    -0.727970
          b     1.102445
          c     1.113834
          d    -0.195407
          dtype: float64
```

```
In [79]: df.reindex(["c", "f", "b"], axis="index")
```

```
Out[79]:
```

	one	two	three	flag
c	-0.367262	1.963174	-0.721000	False
f	NaN	NaN	NaN	NaN
b	0.572949	-0.778513	-0.446048	False

```
In [80]: df.reindex(["three", "two", "one"], axis="columns")
```

```
Out[80]:
```

	three	two	one
a	0.016241	-0.020558	-0.789990
b	-0.446048	-0.778513	0.572949
c	-0.721000	1.963174	-0.367262
d	NaN	-0.313509	NaN

```
In [82]: df.reindex_like(df)
```

```
Out[82]:
```

	one	two	three	flag
a	-0.789990	-0.020558	0.016241	False
b	0.572949	-0.778513	-0.446048	False
c	-0.367262	1.963174	-0.721000	False
d	NaN	-0.313509	NaN	False

```
In [83]: s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
s1 = s[:4]
s2 = s[1:]
s1.align(s2)
```

```
Out[83]: (a    -0.511169
          b    -0.185304
          c     0.725502
          d    -1.852033
          e      NaN
         dtype: float64,
          a      NaN
          b    -0.185304
          c     0.725502
          d    -1.852033
          e    -1.251512
         dtype: float64)
```

```
In [84]: s1.align(s2, join="inner")
```

```
Out[84]: (b    -0.185304
          c    0.725502
          d   -1.852033
         dtype: float64,
          b    -0.185304
          c    0.725502
          d   -1.852033
         dtype: float64)
```

```
In [85]: s1.align(s2, join="left")
```

```
Out[85]: (a    -0.511169
          b    -0.185304
          c    0.725502
          d   -1.852033
         dtype: float64,
          a      NaN
          b    -0.185304
          c    0.725502
          d   -1.852033
         dtype: float64)
```

```
In [88]: df.align(df5, join="inner")
```

```
Out[88]: (Empty DataFrame
          Columns: []
          Index: [],
          Empty DataFrame
          Columns: []
          Index: [])
```

```
In [89]: df.align(df5, join="inner", axis=0)
```

```
Out[89]: (Empty DataFrame
          Columns: [one, two, three, flag]
          Index: [],
          Empty DataFrame
          Columns: [A, B]
          Index: [])
```

```
In [91]: df.align(df5.iloc[0], axis=1)
```

```
Out[91]: (   A    B    flag      one     three      two
       a NaN  NaN  False -0.789990  0.016241 -0.020558
       b NaN  NaN  False  0.572949 -0.446048 -0.778513
       c NaN  NaN  False -0.367262 -0.721000  1.963174
       d NaN  NaN  False        NaN        NaN -0.313509,
          A        0.0
          B        6.0
          flag      NaN
          one      NaN
          three     NaN
          two      NaN
         Name: 0, dtype: float64)
```

```
In [92]: rng = pd.date_range("1/3/2000", periods=8)
ts = pd.Series(np.random.randn(8), index=rng)
ts2 = ts[[0, 3, 6]]
ts
```

```
Out[92]: 2000-01-03    -1.310621
2000-01-04    -0.992201
2000-01-05    -1.394069
2000-01-06     0.820258
2000-01-07    -1.331111
2000-01-08     0.116894
2000-01-09    -0.452949
2000-01-10     1.596265
Freq: D, dtype: float64
```

```
In [93]: ts2.reindex(ts.index)
```

```
Out[93]: 2000-01-03    -1.310621
2000-01-04        NaN
2000-01-05        NaN
2000-01-06     0.820258
2000-01-07        NaN
2000-01-08        NaN
2000-01-09    -0.452949
2000-01-10        NaN
Freq: D, dtype: float64
```

```
In [94]: ts2.reindex(ts.index, method="ffill")
```

```
Out[94]: 2000-01-03    -1.310621
2000-01-04    -1.310621
2000-01-05    -1.310621
2000-01-06     0.820258
2000-01-07     0.820258
2000-01-08     0.820258
2000-01-09    -0.452949
2000-01-10    -0.452949
Freq: D, dtype: float64
```

```
In [95]: ts2.reindex(ts.index, method="bfill")
```

```
Out[95]: 2000-01-03    -1.310621
2000-01-04     0.820258
2000-01-05     0.820258
2000-01-06     0.820258
2000-01-07    -0.452949
2000-01-08    -0.452949
2000-01-09    -0.452949
2000-01-10      NaN
Freq: D, dtype: float64
```

```
In [96]: ts2.reindex(ts.index, method="nearest")
```

```
Out[96]: 2000-01-03    -1.310621
2000-01-04    -1.310621
2000-01-05     0.820258
2000-01-06     0.820258
2000-01-07     0.820258
2000-01-08    -0.452949
2000-01-09    -0.452949
2000-01-10    -0.452949
Freq: D, dtype: float64
```

```
In [97]: ts2.reindex(ts.index).fillna(method="ffill")
```

```
Out[97]: 2000-01-03    -1.310621
2000-01-04    -1.310621
2000-01-05    -1.310621
2000-01-06     0.820258
2000-01-07     0.820258
2000-01-08     0.820258
2000-01-09    -0.452949
2000-01-10    -0.452949
Freq: D, dtype: float64
```

```
In [98]: ts2.reindex(ts.index, method="ffill", limit=1)
```

```
Out[98]: 2000-01-03    -1.310621
2000-01-04    -1.310621
2000-01-05      NaN
2000-01-06     0.820258
2000-01-07     0.820258
2000-01-08      NaN
2000-01-09    -0.452949
2000-01-10    -0.452949
Freq: D, dtype: float64
```

```
In [99]: ts2.reindex(ts.index, method="ffill", tolerance="1 day")
```

```
Out[99]: 2000-01-03    -1.310621
2000-01-04    -1.310621
2000-01-05        NaN
2000-01-06     0.820258
2000-01-07     0.820258
2000-01-08        NaN
2000-01-09   -0.452949
2000-01-10   -0.452949
Freq: D, dtype: float64
```

```
In [100]: df.drop(["a", "d"], axis=0)
```

```
Out[100]:      one      two      three  flag
a  0.572949 -0.778513 -0.446048 False
c -0.367262  1.963174 -0.721000 False
```

```
In [101]: df.drop(["one"], axis=1)
```

```
Out[101]:      two      three  flag
a -0.020558  0.016241 False
b -0.778513 -0.446048 False
c  1.963174 -0.721000 False
d -0.313509      NaN False
```

```
In [102]: df.reindex(df.index.difference(["a", "d"]))
```

```
Out[102]:      one      two      three  flag
b  0.572949 -0.778513 -0.446048 False
c -0.367262  1.963174 -0.721000 False
```

```
In [103]: s.rename(str.upper)
```

```
Out[103]: A    -0.511169
B    -0.185304
C     0.725502
D    -1.852033
E    -1.251512
dtype: float64
```

```
In [104]: df.rename(  
columns={"one": "foo", "two": "bar"},  
index={"a": "apple", "b": "banana", "d": "durian"})
```

```
Out[104]:
```

	foo	bar	three	flag
apple	-0.789990	-0.020558	0.016241	False
banana	0.572949	-0.778513	-0.446048	False
c	-0.367262	1.963174	-0.721000	False
durian		NaN	-0.313509	NaN

```
In [105]: df.rename({"one": "foo", "two": "bar"}, axis="columns")
```

```
Out[105]:
```

	foo	bar	three	flag
a	-0.789990	-0.020558	0.016241	False
b	0.572949	-0.778513	-0.446048	False
c	-0.367262	1.963174	-0.721000	False
d		NaN	-0.313509	NaN

```
In [106]: df.rename({"a": "apple", "b": "banana", "d": "durian"}, axis="index")
```

```
Out[106]:
```

	one	two	three	flag
apple	-0.789990	-0.020558	0.016241	False
banana	0.572949	-0.778513	-0.446048	False
c	-0.367262	1.963174	-0.721000	False
durian		NaN	-0.313509	NaN

```
In [107]: s.rename("scalar-name")
```

```
Out[107]: a    -0.511169  
b    -0.185304  
c     0.725502  
d    -1.852033  
e    -1.251512  
Name: scalar-name, dtype: float64
```

```
In [108]: df = pd.DataFrame(  
    {"x": [1, 2, 3, 4, 5, 6], "y": [10, 20, 30, 40, 50, 60]},  
    index=pd.MultiIndex.from_product(  
        [["a", "b", "c"], [1, 2]], names=["let", "num"]  
    ))  
  
df
```

Out[108]:

	x	y
let	num	
	1	10
a	2	20
	1	30
b	2	40
	1	50
c	2	60

```
In [109]: df.rename_axis(index={"let": "abc"})
```

Out[109]:

	x	y
abc	num	
	1	10
a	2	20
	1	30
b	2	40
	1	50
c	2	60

```
In [110]: df.rename_axis(index=str.upper)
```

Out[110]:

	x	y
LET	NUM	
	1	10
a	2	20
	1	30
b	2	40
	1	50
c	2	60

```
In [111]: df = pd.DataFrame({"col1": np.random.randn(3), "col2": np.random.randn(3)}, index=[0, 1, 2])

for col in df:
    print(col)
```

```
col1
col2
```

```
In [112]: df = pd.DataFrame({"a": [1, 2, 3], "b": ["a", "b", "c"]})
In [257]: for index, row in df.iterrows():
            row["a"] = 10
df
```

```
Out[112]:   a   b
              0   a
              1   b
              2   c
```

```
In [113]: for label, ser in df.items():
            print(label)
            print(ser)
```

```
a
0    1
1    2
2    3
Name: a, dtype: int64
b
0    a
1    b
2    c
Name: b, dtype: object
```

```
In [114]: for row_index, row in df.iterrows():
            print(row_index, row, sep="\n")
```

```
0
a    1
b    a
Name: 0, dtype: object
1
a    2
b    b
Name: 1, dtype: object
2
a    3
b    c
Name: 2, dtype: object
```

```
In [115]: df_orig = pd.DataFrame([[1, 1.5]], columns=["int", "float"])
df_orig.dtypes
```

```
Out[115]: int      int64
          float    float64
          dtype: object
```

```
In [116]: row = next(df_orig.iterrows())[1]
row
```

```
Out[116]: int      1.0
          float    1.5
          Name: 0, dtype: float64
```

```
In [117]: row["int"].dtype
```

```
Out[117]: dtype('float64')
```

```
In [118]: df_orig["int"].dtype
```

```
Out[118]: dtype('int64')
```

```
In [119]: df2 = pd.DataFrame({"x": [1, 2, 3], "y": [4, 5, 6]})
print(df2)
```

```
      x  y
0   1  4
1   2  5
2   3  6
```

```
In [120]: df2_t = pd.DataFrame({idx: values for idx, values in df2.iterrows()})
print(df2_t)
```

```
      0  1  2
x  1  2  3
y  4  5  6
```

```
In [121]: for row in df.itertuples():
           print(row)
```

```
Pandas(Index=0, a=1, b='a')
Pandas(Index=1, a=2, b='b')
Pandas(Index=2, a=3, b='c')
```

```
In [122]: s = pd.Series(pd.date_range("20130101 09:10:12", periods=4))
s
```

```
Out[122]: 0    2013-01-01 09:10:12
1    2013-01-02 09:10:12
2    2013-01-03 09:10:12
3    2013-01-04 09:10:12
dtype: datetime64[ns]
```

```
In [123]: s.dt.hour
```

```
Out[123]: 0    9  
1    9  
2    9  
3    9  
dtype: int64
```

```
In [124]: s.dt.second
```

```
Out[124]: 0    12  
1    12  
2    12  
3    12  
dtype: int64
```

```
In [125]: s.dt.day
```

```
Out[125]: 0    1  
1    2  
2    3  
3    4  
dtype: int64
```

```
In [126]: s[s.dt.day == 2]
```

```
Out[126]: 1    2013-01-02 09:10:12  
dtype: datetime64[ns]
```

```
In [127]: stz = s.dt.tz_localize("US/Eastern")  
stz
```

```
Out[127]: 0    2013-01-01 09:10:12-05:00  
1    2013-01-02 09:10:12-05:00  
2    2013-01-03 09:10:12-05:00  
3    2013-01-04 09:10:12-05:00  
dtype: datetime64[ns, US/Eastern]
```

```
In [128]: s.dt.tz_localize("UTC").dt.tz_convert("US/Eastern")
```

```
Out[128]: 0    2013-01-01 04:10:12-05:00  
1    2013-01-02 04:10:12-05:00  
2    2013-01-03 04:10:12-05:00  
3    2013-01-04 04:10:12-05:00  
dtype: datetime64[ns, US/Eastern]
```

```
In [129]: s = pd.Series(pd.date_range("20130101", periods=4))
s
```

```
Out[129]: 0    2013-01-01
1    2013-01-02
2    2013-01-03
3    2013-01-04
dtype: datetime64[ns]
```

```
In [130]: s.dt.strftime("%Y/%m/%d")
```

```
Out[130]: 0    2013/01/01
1    2013/01/02
2    2013/01/03
3    2013/01/04
dtype: object
```

```
In [132]: s = pd.Series(pd.period_range("20130101", periods=4))
s
```

```
Out[132]: 0    2013-01-01
1    2013-01-02
2    2013-01-03
3    2013-01-04
dtype: period[D]
```

```
In [133]: s.dt.strftime("%Y/%m/%d")
```

```
Out[133]: 0    2013/01/01
1    2013/01/02
2    2013/01/03
3    2013/01/04
dtype: object
```

```
In [134]: s = pd.Series(pd.period_range("20130101", periods=4, freq="D"))
s
```

```
Out[134]: 0    2013-01-01
1    2013-01-02
2    2013-01-03
3    2013-01-04
dtype: period[D]
```

```
In [135]: s.dt.year
```

```
Out[135]: 0    2013
1    2013
2    2013
3    2013
dtype: int64
```

```
In [136]: s.dt.day
```

```
Out[136]: 0    1  
1    2  
2    3  
3    4  
dtype: int64
```

```
In [137]: s = pd.Series(pd.timedelta_range("1 day 00:00:05", periods=4, freq="s"))  
s
```

```
Out[137]: 0    1 days 00:00:05  
1    1 days 00:00:06  
2    1 days 00:00:07  
3    1 days 00:00:08  
dtype: timedelta64[ns]
```

```
In [138]: s.dt.days
```

```
Out[138]: 0    1  
1    1  
2    1  
3    1  
dtype: int64
```

```
In [139]: s.dt.seconds
```

```
Out[139]: 0    5  
1    6  
2    7  
3    8  
dtype: int64
```

```
In [140]: s.dt.components
```

	days	hours	minutes	seconds	milliseconds	microseconds	nanoseconds
0	1	0	0	5	0	0	0
1	1	0	0	6	0	0	0
2	1	0	0	7	0	0	0
3	1	0	0	8	0	0	0


```
In [1]: import pandas as pd  
import numpy as np
```

```
In [2]: df = pd.DataFrame(  
{  
    "one": pd.Series(np.random.randn(3), index=["a", "b", "c"]),  
    "two": pd.Series(np.random.randn(4), index=["a", "b", "c", "d"]),  
    "three": pd.Series(np.random.randn(3), index=["b", "c", "d"]),  
}  
)  
unsorted_df = df.reindex(  
    index=["a", "d", "c", "b"], columns=["three", "two", "one"]  
)  
  
unsorted_df
```

```
Out[2]:   three      two      one  
            
a      NaN -0.326406 -0.078773  
d  1.122434 -0.263896      NaN  
c  0.745061 -0.700178  2.138660  
b -2.535724 -0.196084  1.556742
```

```
In [3]: unsorted_df.sort_index()
```

```
Out[3]:   three      two      one  
            
a      NaN -0.326406 -0.078773  
b -2.535724 -0.196084  1.556742  
c  0.745061 -0.700178  2.138660  
d  1.122434 -0.263896      NaN
```

```
In [4]: unsorted_df.sort_index(ascending=False)
```

```
Out[4]:   three      two      one  
            
d  1.122434 -0.263896      NaN  
c  0.745061 -0.700178  2.138660  
b -2.535724 -0.196084  1.556742  
a      NaN -0.326406 -0.078773
```

```
In [5]: unsorted_df.sort_index(axis=1)
```

```
Out[5]:      one     three     two
a -0.078773      NaN -0.326406
d      NaN  1.122434 -0.263896
c  2.138660  0.745061 -0.700178
b  1.556742 -2.535724 -0.196084
```

```
In [6]: unsorted_df["three"].sort_index()
```

```
Out[6]: a      NaN
b -2.535724
c 0.745061
d 1.122434
Name: three, dtype: float64
```

```
In [7]: s1 = pd.DataFrame({"a": ["B", "a", "C"], "b": [1, 2, 3], "c": [2, 3, 4]}).set_index("a")
s1
```

```
Out[7]:      c
      a   b
      a   b
      B   1   2
      a   2   3
      C   3   4
```

```
In [8]: s1.sort_index(level="a")
```

```
Out[8]:      c
      a   b
      a   b
      B   1   2
      C   3   4
      a   2   3
```

```
In [9]: s1.sort_index(level="a", key=lambda idx: idx.str.lower())
```

```
Out[9]:      c
      a   b
      a   b
      a   2   3
      B   1   2
      C   3   4
```

```
In [10]: df1 = pd.DataFrame({"one": [2, 1, 1, 1], "two": [1, 3, 2, 4], "three": [5, 4, 3, 2]})

df1.sort_values(by="two")
```

```
Out[10]:   one  two  three
0      2    1      5
2      1    2      3
1      1    3      4
3      1    4      2
```

```
In [11]: df1[["one", "two", "three"]].sort_values(by=["one", "two"])
```

```
Out[11]:   one  two  three
2      1    2      3
1      1    3      4
3      1    4      2
0      2    1      5
```

```
In [13]: s = pd.Series(np.random.randn(5), index=["a", "b", "c", "d", "e"])
s
```

```
Out[13]: a    -0.205803
b     0.766681
c     0.095059
d     1.604067
e     0.372262
dtype: float64
```

```
In [14]: s[2] = np.nan
s.sort_values()
```

```
Out[14]: a    -0.205803
e     0.372262
b     0.766681
d     1.604067
c      NaN
dtype: float64
```

```
In [15]: s.sort_values(na_position="first")
```

```
Out[15]: c      NaN
a    -0.205803
e     0.372262
b     0.766681
d     1.604067
dtype: float64
```

```
In [16]: s1 = pd.Series(["B", "a", "C"])
s1.sort_values()
```

```
Out[16]: 0      B
          2      C
          1      a
          dtype: object
```

```
In [17]: df = pd.DataFrame({"a": ["B", "a", "C"], "b": [1, 2, 3]})
df.sort_values(by="a")
```

```
Out[17]:    a   b
0   B   1
2   C   3
1   a   2
```

```
In [18]: df.sort_values(by="a", key=lambda col: col.str.lower())
```

```
Out[18]:    a   b
1   a   2
0   B   1
2   C   3
```

```
In [19]: idx = pd.MultiIndex.from_tuples([('a', 1), ('a', 2), ('a', 2), ('b', 2), ('b', 2),
idx.names = ["first", "second"]
df_multi = pd.DataFrame({"A": np.arange(6, 0, -1)}, index=idx)
df_multi
```

```
Out[19]:      A
              first  second
                1       6
              a        2       5
                  2       4
                  2       3
              b        1       2
                  1       1
```

```
In [21]: df_multi.sort_values(by=["second", "A"])
```

```
Out[21]: A
```

	first	second
b	1	1
	1	2
a	1	6
b	2	3
	2	4
a	2	5

```
In [22]: ser = pd.Series([1, 2, 3])
ser.searchsorted([0, 3])
```

```
Out[22]: array([0, 2], dtype=int64)
```

```
In [23]: ser.searchsorted([0, 4])
```

```
Out[23]: array([0, 3], dtype=int64)
```

```
In [24]: ser.searchsorted([0, 3], sorter=np.argsort(ser))
```

```
Out[24]: array([0, 2], dtype=int64)
```

```
In [25]: s = pd.Series(np.random.permutation(10))
s
```

```
Out[25]: 0    4
1    7
2    1
3    2
4    0
5    8
6    5
7    6
8    9
9    3
dtype: int32
```

```
In [26]: df = pd.DataFrame({
    "a": [-2, -1, 1, 10, 8, 11, -1],
    "b": list("abdceff"),
    "c": [1.0, 2.0, 4.0, 3.2, np.nan, 3.0, 4.0],
})

df.nlargest(3, "a")
```

```
Out[26]:   a   b     c
      5   f   3.0
      3   10  c   3.2
      4   8   e   NaN
```

```
In [27]: df1.columns = pd.MultiIndex.from_tuples([('a', "one"), ("a", "two"), ("b", "three")])

df1.sort_values(by=("a", "two"))
```

```
Out[27]:      a      b
          one  two  three
      0   2   1   5
      2   1   2   3
      1   1   3   4
      3   1   4   2
```

```
In [28]: dft = pd.DataFrame({
    "A": np.random.rand(3),
    "B": 1,
    "C": "foo",
    "D": pd.Timestamp("20010102"),
    "E": pd.Series([1.0] * 3).astype("float32"),
    "F": False,
    "G": pd.Series([1] * 3, dtype="int8"),
})

dft
```

```
Out[28]:      A      B      C          D      E      F      G
      0  0.577873  1  foo  2001-01-02  1.0  False  1
      1  0.149990  1  foo  2001-01-02  1.0  False  1
      2  0.244930  1  foo  2001-01-02  1.0  False  1
```

```
In [29]: df1 = pd.DataFrame(np.random.randn(8, 1), columns=["A"], dtype="float32")
df1
```

```
Out[29]:      A
_____
0    0.466913
1    1.734496
2    0.416978
3    0.158830
4    0.626867
5   -1.188689
6   -2.190499
7   -0.572933
```

```
In [30]: df2 = pd.DataFrame({
    "A": pd.Series(np.random.randn(8), dtype="float16"),
    "B": pd.Series(np.random.randn(8)),
    "C": pd.Series(np.array(np.random.randn(8)), dtype="uint8")),
})

df2
```

```
Out[30]:      A        B    C
_____
0  0.487549  0.544818  0
1  1.299805  0.228472  0
2  0.007591  0.703416  0
3  0.010628  0.621133  0
4  1.896484 -1.264181  0
5  1.053711 -0.364295  255
6  0.562988 -0.390742  0
7  0.713379  0.547247  0
```

```
In [31]: pd.DataFrame([1, 2], columns=["a"]).dtypes
```

```
Out[31]: a    int64
dtype: object
```

```
In [32]: pd.DataFrame({"a": [1, 2]}).dtypes
```

```
Out[32]: a    int64
dtype: object
```

```
In [33]: pd.DataFrame({"a": 1}, index=list(range(2))).dtypes
```

```
Out[33]: a    int64
dtype: object
```

```
In [34]: df3 = df1.reindex_like(df2).fillna(value=0.0) + df2  
df3
```

```
Out[34]:
```

	A	B	C
0	0.954462	0.544818	0.0
1	3.034301	0.228472	0.0
2	0.424570	0.703416	0.0
3	0.169458	0.621133	0.0
4	2.523352	-1.264181	0.0
5	-0.134978	-0.364295	255.0
6	-1.627511	-0.390742	0.0
7	0.140446	0.547247	0.0

```
In [35]: df3.to_numpy().dtype
```

```
Out[35]: dtype('float64')
```

```
In [36]: df3.astype("float32").dtypes
```

```
Out[36]: A      float32  
B      float32  
C      float32  
dtype: object
```

```
In [37]: dft = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6], "c": [7, 8, 9]})  
dft[["a", "b"]] = dft[["a", "b"]].astype(np.uint8)  
dft
```

```
Out[37]:
```

	a	b	c
0	1	4	7
1	2	5	8
2	3	6	9

```
In [38]: dft1 = pd.DataFrame({"a": [1, 0, 1], "b": [4, 5, 6], "c": [7, 8, 9]})  
dft1 = dft1.astype({"a": np.bool_, "c": np.float64})  
dft1
```

```
Out[38]:
```

	a	b	c
0	True	4	7.0
1	False	5	8.0
2	True	6	9.0

```
In [39]: dft = pd.DataFrame({"a": [1, 2, 3], "b": [4, 5, 6], "c": [7, 8, 9]})  
dft.loc[:, ["a", "b"]].astype(np.uint8).dtypes
```

```
Out[39]: a    uint8  
b    uint8  
dtype: object
```

```
In [40]: dft.loc[:, ["a", "b"]] = dft.loc[:, ["a", "b"]].astype(np.uint8)  
dft.dtypes
```

```
Out[40]: a    int64  
b    int64  
c    int64  
dtype: object
```

```
In [41]: import datetime  
df = pd.DataFrame(  
[  
[1, 2],  
["a", "b"],  
[datetime.datetime(2016, 3, 2), datetime.datetime(2016, 3, 2)],  
]  
)  
df = df.T  
df
```

```
Out[41]:      0      1      2  
0   1   a  2016-03-02  
1   2   b  2016-03-02
```

```
In [42]: df.infer_objects().dtypes
```

```
Out[42]: 0           int64  
1           object  
2  datetime64[ns]  
dtype: object
```

```
In [ ]: m = ["1.1", 2, 3]  
pd.to_numeric(m)
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [43]: import datetime
m = ["2016-07-09", datetime.datetime(2016, 3, 2)]
pd.to_datetime(m)
```

```
Out[43]: DatetimeIndex(['2016-07-09', '2016-03-02'], dtype='datetime64[ns]', freq=None)
```

```
In [44]: m = ["5us", pd.Timedelta("1day")]
pd.to_timedelta(m)
```

```
Out[44]: TimedeltaIndex(['0 days 00:00:00.000005', '1 days 00:00:00'], dtype='timedelta64[ns]', freq=None)
```

```
In [45]: import datetime
m = ["apple", datetime.datetime(2016, 3, 2)]
```

```
In [46]: pd.to_datetime(m, errors="coerce")
```

```
Out[46]: DatetimeIndex(['NaT', '2016-03-02'], dtype='datetime64[ns]', freq=None)
```

```
In [47]: m = ["apple", 2, 3]
pd.to_numeric(m, errors="coerce")
```

```
Out[47]: array([nan, 2., 3.])
```

```
In [48]: m = ["apple", pd.Timedelta("1day")]
pd.to_timedelta(m, errors="coerce")
```

```
Out[48]: TimedeltaIndex(['NaT', '1 days'], dtype='timedelta64[ns]', freq=None)
```

```
In [49]: import datetime
m = ["apple", datetime.datetime(2016, 3, 2)]
pd.to_datetime(m, errors="ignore")
```

```
Out[49]: Index(['apple', 2016-03-02 00:00:00], dtype='object')
```

```
In [50]: m = ["apple", 2, 3]
pd.to_numeric(m, errors="ignore")
```

```
Out[50]: array(['apple', 2, 3], dtype=object)
```

```
In [51]: m = ["apple", pd.Timedelta("1day")]
pd.to_timedelta(m, errors="ignore")
```

```
Out[51]: array(['apple', Timedelta('1 days 00:00:00')], dtype=object)
```

```
In [52]: import datetime  
df = pd.DataFrame([["2016-07-09", datetime.datetime(2016, 3, 2)] * 2, dtype="O")  
df
```

```
Out[52]:      0           1  
0  2016-07-09  2016-03-02 00:00:00  
1  2016-07-09  2016-03-02 00:00:00
```

```
In [54]: df.apply(pd.to_datetime)
```

```
Out[54]:      0           1  
0  2016-07-09  2016-03-02  
1  2016-07-09  2016-03-02
```

```
In [55]: df = pd.DataFrame([[1.1, 2, 3]] * 2, dtype="O")  
df
```

```
Out[55]:      0   1   2  
0  1.1  2  3  
1  1.1  2  3
```

```
In [56]: df.apply(pd.to_numeric)
```

```
Out[56]:      0   1   2  
0  1.1  2  3  
1  1.1  2  3
```

```
In [57]: df = pd.DataFrame([["5us", pd.Timedelta("1day")]] * 2, dtype="O")  
df
```

```
Out[57]:      0           1  
0  5us  1 days 00:00:00  
1  5us  1 days 00:00:00
```

```
In [58]: df.apply(pd.to_timedelta)
```

```
Out[58]:      0           1  
0  0 days 00:00:00.000005  1 days  
1  0 days 00:00:00.000005  1 days
```

```
In [59]: dfi = df3.astype("int32")
dfi["E"] = 1
dfi
```

```
Out[59]:   A  B  C  E
0  0  0  0  1
1  3  0  0  1
2  0  0  0  1
3  0  0  0  1
4  2 -1  0  1
5  0  0  255 1
6 -1  0  0  1
7  0  0  0  1
```

```
In [60]: casted = dfi[dfi > 0]
casted
```

```
Out[60]:   A  B  C  E
0  NaN  NaN  NaN  1
1  3.0  NaN  NaN  1
2  NaN  NaN  NaN  1
3  NaN  NaN  NaN  1
4  2.0  NaN  NaN  1
5  NaN  NaN  255.0 1
6  NaN  NaN  NaN  1
7  NaN  NaN  NaN  1
```

```
In [61]: df = pd.DataFrame({
    "string": list("abc"),
    "int64": list(range(1, 4)),
    "uint8": np.arange(3, 6).astype("u1"),
    "float64": np.arange(4.0, 7.0),
    "bool1": [True, False, True],
    "bool2": [False, True, False],
    "dates": pd.date_range("now", periods=3),
    "category": pd.Series(list("ABC")).astype("category"),
})
```

```
In [62]: df["tdeltas"] = df.dates.diff()
df["uint64"] = np.arange(3, 6).astype("u8")
df["other_dates"] = pd.date_range("20130101", periods=3)
df["tz_aware_dates"] = pd.date_range("20130101", periods=3, tz="US/Eastern")
df
```

```
Out[62]:   string  int64  uint8  float64  bool1  bool2          dates  category  tdeltas  uint64  other_dats
0      a      1      3      4.0   True  False  2022-09-22 19:45:48.623494      A      NaT      3  2013-01-
1      b      2      4      5.0  False   True  2022-09-23 19:45:48.623494      B  1 days      4  2013-01-
2      c      3      5      6.0   True  False  2022-09-24 19:45:48.623494      C  1 days      5  2013-01-
```

```
In [63]: df.select_dtypes(include=[bool])
```

```
Out[63]:   bool1  bool2
0    True  False
1  False   True
2    True  False
```

```
In [65]: df.select_dtypes(include=["bool"])
```

```
Out[65]:   bool1  bool2
0    True  False
1  False   True
2    True  False
```

```
In [66]: df.select_dtypes(include=["number", "bool"], exclude=["unsignedinteger"])
```

```
Out[66]:   int64  float64  bool1  bool2  tdeltas
0      1      4.0   True  False      NaT
1      2      5.0  False   True  1 days
2      3      6.0   True  False  1 days
```

```
In [67]: df.select_dtypes(include=["object"])
```

```
Out[67]:   string
0      a
1      b
2      c
```

```
In [68]: def subdtypes(dtype):
    subs = dtype.__subclasses__()
    if not subs:
        return dtype
    return [dtype, [subdtypes(dt) for dt in subs]]
```

```
In [69]: subdtypes(np.generic)
```

```
Out[69]: [numpy.generic,
[[numpy.number,
[[numpy.integer,
[[numpy.signedinteger,
[numpy.int8,
numpy.int16,
numpy.intc,
numpy.int32,
numpy.int64,
numpy.timedelta64]],
[numpy.unsignedinteger,
[numpy.uint8, numpy.uint16, numpy.uintc, numpy.uint32, numpy.uint64]]],
[numpy.inexact,
[[numpy.floating,
[numpy.float16, numpy.float32, numpy.float64, numpy.longdouble]],
[numpy.complexfloating,
[numpy.complex64, numpy.complex128, numpy.clongdouble]]]]],
[numpy.flexible,
[[numpy.character, [numpy.bytes_, numpy.str_]],
[numpy.void, [numpy.record]]]],
numpy.bool_,
numpy.datetime64,
numpy.object_]]]
```

```
In [70]: import pandas as pd
from io import StringIO
data = "col1,col2,col3\na,b,1\na,b,2\nnc,d,3"
pd.read_csv(StringIO(data))
```

```
Out[70]:   col1  col2  col3
0      a      b      1
1      a      b      2
2      c      d      3
```

```
In [71]: pd.read_csv(StringIO(data), usecols=lambda x: x.upper() in ["COL1", "COL3"])
```

```
Out[71]:   col1  col3
0      a      1
1      a      2
2      c      3
```

```
In [72]: data = "col1,col2,col3\na,b,1"
df = pd.read_csv(StringIO(data))
```

```
In [73]: df.columns = [f"pre_{col}" for col in df.columns]
df
```

```
Out[73]:      pre_col1  pre_col2  pre_col3
              0          a          b          1
```

```
In [74]: data = "col1,col2,col3\na,b,1\na,b,2\nnc,d,3"
pd.read_csv(StringIO(data))
```

```
Out[74]:      col1  col2  col3
              0      a      b      1
              1      a      b      2
              2      c      d      3
```

```
In [75]: pd.read_csv(StringIO(data), skiprows=lambda x: x % 2 != 0)
```

```
Out[75]:      col1  col2  col3
              0      a      b      2
```

```
In [76]: import numpy as np
data = "a,b,c,d\n1,2,3,4\n5,6,7,8\n9,10,11"
print(data)
```

```
a,b,c,d
1,2,3,4
5,6,7,8
9,10,11
```

```
In [77]: df = pd.read_csv(StringIO(data), dtype=object)
df
```

```
Out[77]:      a    b    c    d
              0    1    2    3    4
              1    5    6    7    8
              2    9   10   11   NaN
```

```
In [78]: df = pd.read_csv(StringIO(data), dtype={"b": object, "c": np.float64, "d": "Int64"})
df.dtypes
```

```
Out[78]: a      int64
         b      object
         c    float64
         d      Int64
dtype: object
```

```
In [79]: data = "col_1\n1\n2\n'A'\n4.22"
pd.read_csv(StringIO(data), converters={"col_1": str})
df
```

```
Out[79]:
```

	a	b	c	d
0	1	2	3.0	4
1	5	6	7.0	8
2	9	10	11.0	<NA>

```
In [80]: df = pd.read_csv(StringIO(data), dtype="category")
df.dtypes
```

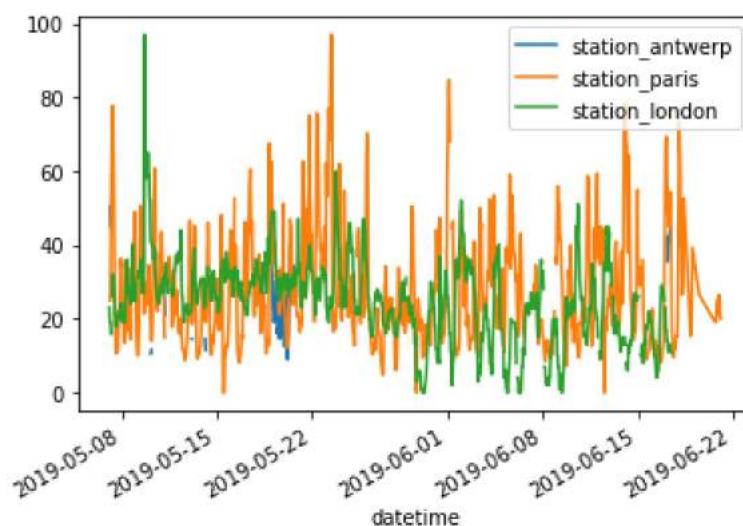
```
Out[80]: col_1    category
dtype: object
```

```
In [81]: air_quality = pd.read_csv("air_quality_no2.csv", index_col=0, parse_dates=True)
air_quality.head()
```

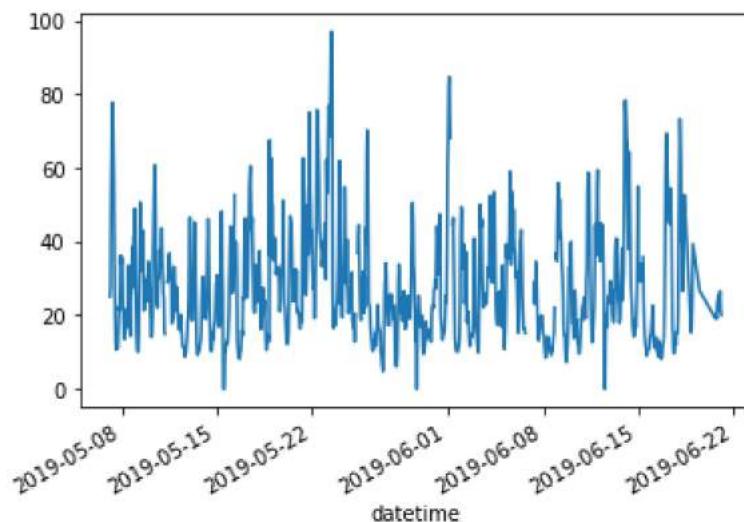
```
Out[81]:
```

	station_antwerp	station_paris	station_london
datetime			
2019-05-07 02:00:00	NaN	NaN	23.0
2019-05-07 03:00:00	50.5	25.0	19.0
2019-05-07 04:00:00	45.0	27.7	19.0
2019-05-07 05:00:00	NaN	50.4	16.0
2019-05-07 06:00:00	NaN	61.9	NaN

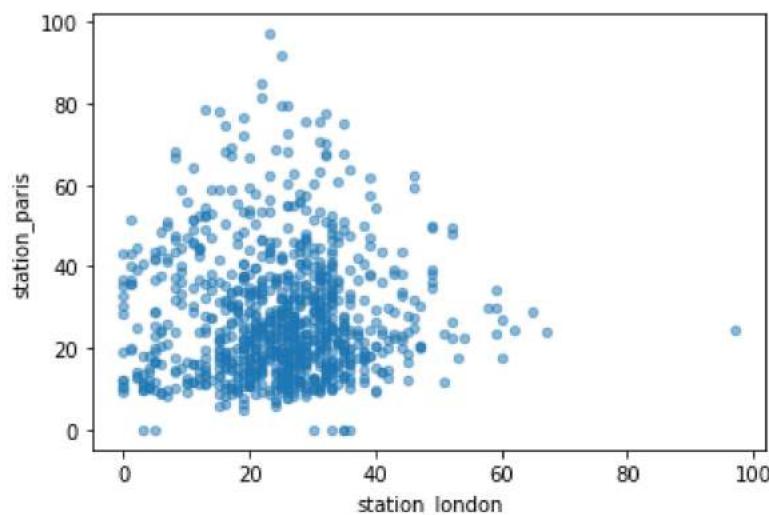
```
In [82]: air_quality.plot();
```



```
In [83]: air_quality["station_paris"].plot();
```



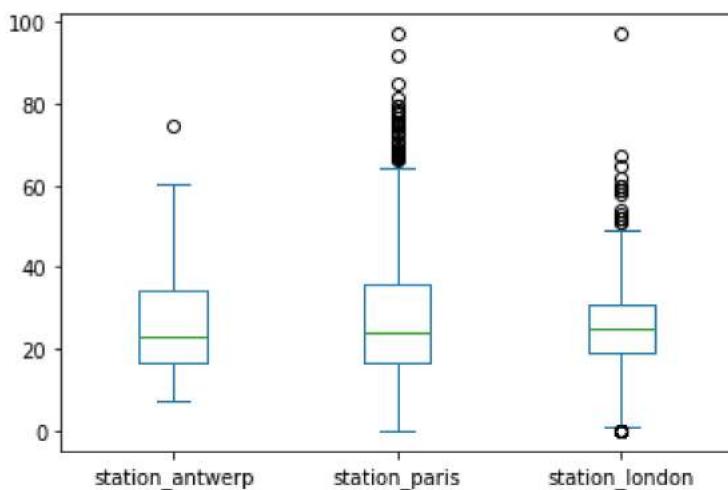
```
In [84]: air_quality.plot.scatter(x="station_london", y="station_paris", alpha=0.5);
```



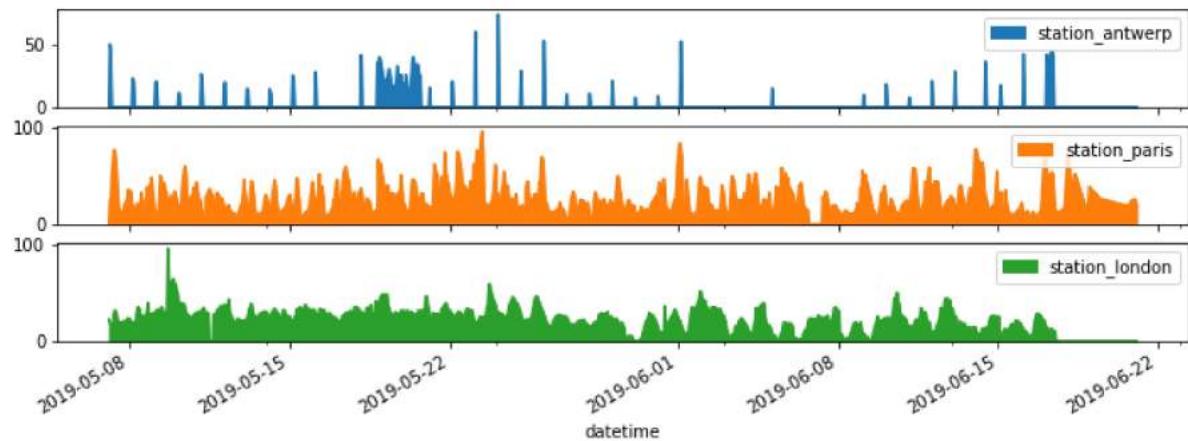
```
In [85]: [  
    method_name  
    for method_name in dir(air_quality.plot)  
    if not method_name.startswith("_")  
]
```

```
Out[85]: ['area',  
         'bar',  
         'barh',  
         'box',  
         'density',  
         'hexbin',  
         'hist',  
         'kde',  
         'line',  
         'pie',  
         'scatter']
```

```
In [86]: air_quality.plot.box();
```

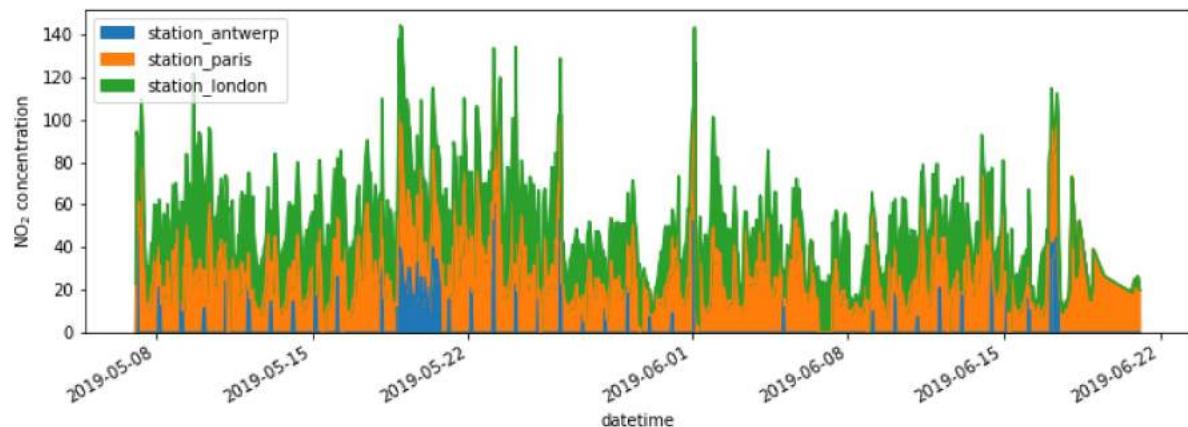


```
In [87]: axs = air_quality.plot.area(figsize=(12, 4), subplots=True);  
axs;
```



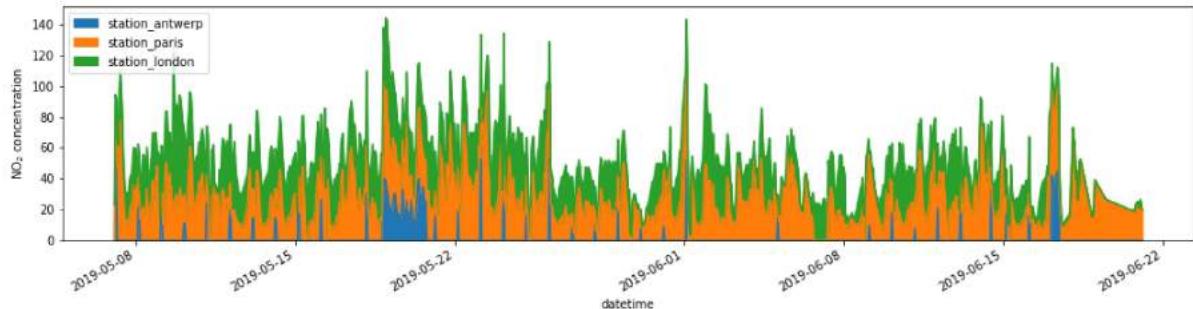
```
In [88]: fig, axs = plt.subplots(figsize=(12, 4))  
air_quality.plot.area(ax=axs)  
axs.set_ylabel("NO$ _2 $ concentration")  
fig.savefig("no2_concentrations.png")
```

<IPython.core.display.Javascript object>



```
In [89]: fig, axs = plt.subplots(figsize=(17, 4))
air_quality.plot.area(ax=axs)
axs.set_ylabel("NO$₂$ concentration");
```

<IPython.core.display.Javascript object>



```
In [90]: air_quality["london_mg_per_cubic"] = air_quality["station_london"] * 1.882
air_quality.head()
```

Out[90]:

	station_antwerp	station_paris	station_london	london_mg_per_cubic
datetime				
2019-05-07 02:00:00		NaN	NaN	23.0
2019-05-07 03:00:00		50.5	25.0	35.758
2019-05-07 04:00:00		45.0	27.7	35.758
2019-05-07 05:00:00		NaN	50.4	30.112
2019-05-07 06:00:00		NaN	61.9	NaN

```
In [91]: air_quality["ratio_paris_antwerp"] = (air_quality["station_paris"] / air_quality["station_antwerp"])
air_quality.head()
```

Out[91]:

	station_antwerp	station_paris	station_london	london_mg_per_cubic	ratio_paris_antwerp
datetime					
2019-05-07 02:00:00		NaN	NaN	23.0	43.286
2019-05-07 03:00:00		50.5	25.0	19.0	35.758
2019-05-07 04:00:00		45.0	27.7	19.0	35.758
2019-05-07 05:00:00		NaN	50.4	16.0	30.112
2019-05-07 06:00:00		NaN	61.9	NaN	NaN

```
In [92]: air_quality_renamed = air_quality.rename(
    columns={
        "station_antwerp": "BETR801",
        "station_paris": "FR04014",
        "station_london": "London Westminster",
    }
)

air_quality_renamed.head()
```

Out[92]:

	BETR801	FR04014	London Westminster	london_mg_per_cubic	ratio_paris_antwerp
datetime					
2019-05-07 02:00:00	NaN	NaN	23.0	43.286	NaN
2019-05-07 03:00:00	50.5	25.0	19.0	35.758	0.495050
2019-05-07 04:00:00	45.0	27.7	19.0	35.758	0.615556
2019-05-07 05:00:00	NaN	50.4	16.0	30.112	NaN
2019-05-07 06:00:00	NaN	61.9	NaN	NaN	NaN

```
In [97]: air_quality_renamed = air_quality_renamed.rename(columns=str.lower)
air_quality_renamed.head()
```

Out[97]:

	betr801	fr04014	london westminster	london_mg_per_cubic	ratio_paris_antwerp
datetime					
2019-05-07 02:00:00	NaN	NaN	23.0	43.286	NaN
2019-05-07 03:00:00	50.5	25.0	19.0	35.758	0.495050
2019-05-07 04:00:00	45.0	27.7	19.0	35.758	0.615556
2019-05-07 05:00:00	NaN	50.4	16.0	30.112	NaN
2019-05-07 06:00:00	NaN	61.9	NaN	NaN	NaN

```
In [99]: air_quality = pd.read_csv("air_quality_long.csv", index_col="date.utc", parse_date=True)
air_quality.head()
```

```
Out[99]:
```

	city	country	location	parameter	value	unit
date.utc						
2019-06-18 06:00:00+00:00	Antwerpen	BE	BETR801	pm25	18.0	µg/m³
2019-06-17 08:00:00+00:00	Antwerpen	BE	BETR801	pm25	6.5	µg/m³
2019-06-17 07:00:00+00:00	Antwerpen	BE	BETR801	pm25	18.5	µg/m³
2019-06-17 06:00:00+00:00	Antwerpen	BE	BETR801	pm25	16.0	µg/m³
2019-06-17 05:00:00+00:00	Antwerpen	BE	BETR801	pm25	7.5	µg/m³

```
In [100]: no2 = air_quality[air_quality["parameter"] == "no2"]
```

```
In [101]: no2_subset = no2.sort_index().groupby(["location"]).head(2)
no2_subset
```

```
Out[101]:
```

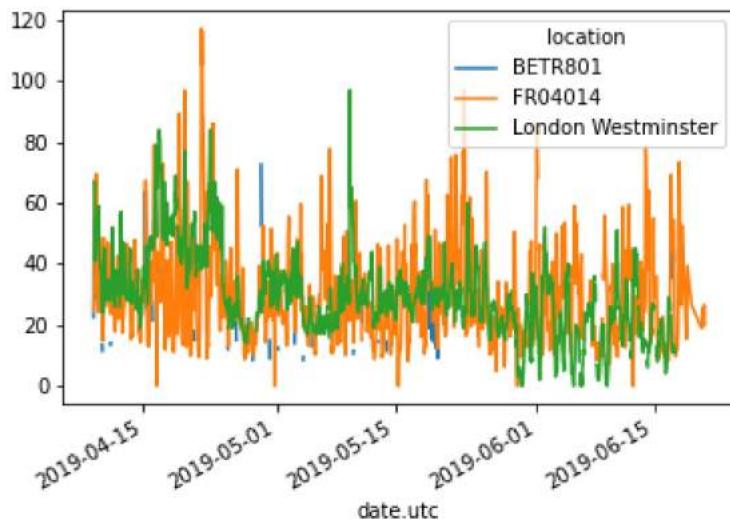
	city	country	location	parameter	value	unit
date.utc						
2019-04-09 01:00:00+00:00	Antwerpen	BE	BETR801	no2	22.5	µg/m³
2019-04-09 01:00:00+00:00	Paris	FR	FR04014	no2	24.4	µg/m³
2019-04-09 02:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³
2019-04-09 02:00:00+00:00	Antwerpen	BE	BETR801	no2	53.5	µg/m³
2019-04-09 02:00:00+00:00	Paris	FR	FR04014	no2	27.4	µg/m³
2019-04-09 03:00:00+00:00	London	GB	London Westminster	no2	67.0	µg/m³

```
In [102]: no2_subset.pivot(columns="location", values="value")
```

```
Out[102]:
```

	location	BETR801	FR04014	London Westminster
date.utc				
2019-04-09 01:00:00+00:00	22.5	24.4	NaN	
2019-04-09 02:00:00+00:00	53.5	27.4	67.0	
2019-04-09 03:00:00+00:00	NaN	NaN	67.0	

```
In [103]: no2.pivot(columns="location", values="value").plot();
```



```
In [104]: air_quality.pivot_table(values="value", index="location", columns="parameter",
```

```
Out[104]:
```

location	parameter	no2	pm25
BETR801	26.950920	23.169492	
FR04014	29.374284		NaN
London Westminster	29.740050	13.443568	

```
In [105]: air_quality.pivot_table(  
values="value",  
index="location",  
columns="parameter",  
aggfunc="mean",  
margins=True,  
)
```

```
Out[105]:
```

location	parameter	no2	pm25	All
BETR801	26.950920	23.169492	24.982353	
FR04014	29.374284		NaN	29.374284
London Westminster	29.740050	13.443568	21.491708	
All	29.430316	14.386849	24.222743	

```
In [106]: air_quality.groupby(["parameter", "location"]).mean()
```

```
Out[106]:
```

parameter	location	value
	BETR801	26.950920
no2	FR04014	29.374284
	London Westminster	29.740050
	BETR801	23.169492
pm25	London Westminster	13.443568

```
In [107]: no2_pivoted = no2.pivot(columns="location", values="value").reset_index()  
no2_pivoted.head()
```

```
Out[107]:
```

	location	date.utc	BETR801	FR04014	London Westminster
0		2019-04-09 01:00:00+00:00	22.5	24.4	NaN
1		2019-04-09 02:00:00+00:00	53.5	27.4	67.0
2		2019-04-09 03:00:00+00:00	54.5	34.2	67.0
3		2019-04-09 04:00:00+00:00	34.5	48.5	41.0
4		2019-04-09 05:00:00+00:00	46.5	59.5	41.0

```
In [108]: no_2 = no2_pivoted.melt(id_vars="date.utc")  
no_2.head()
```

```
Out[108]:
```

	date.utc	location	value
0	2019-04-09 01:00:00+00:00	BETR801	22.5
1	2019-04-09 02:00:00+00:00	BETR801	53.5
2	2019-04-09 03:00:00+00:00	BETR801	54.5
3	2019-04-09 04:00:00+00:00	BETR801	34.5
4	2019-04-09 05:00:00+00:00	BETR801	46.5

```
In [109]: no_2 = no2_pivoted.melt(
    id_vars="date.utc",
    value_vars=["BETR801", "FR04014", "London Westminster"],
    value_name="NO_2",
    var_name="id_location",
)

no_2.head()
```

Out[109]:

	date.utc	id_location	NO_2
0	2019-04-09 01:00:00+00:00	BETR801	22.5
1	2019-04-09 02:00:00+00:00	BETR801	53.5
2	2019-04-09 03:00:00+00:00	BETR801	54.5
3	2019-04-09 04:00:00+00:00	BETR801	34.5
4	2019-04-09 05:00:00+00:00	BETR801	46.5

```
In [110]: air_quality_no2 = pd.read_csv("air_quality_no2_long.csv",parse_dates=True)
air_quality_no2 = air_quality_no2[["date.utc", "location", "parameter", "value"]
air_quality_no2.head()
```

Out[110]:

	date.utc	location	parameter	value
0	2019-06-21 00:00:00+00:00	FR04014	no2	20.0
1	2019-06-20 23:00:00+00:00	FR04014	no2	21.8
2	2019-06-20 22:00:00+00:00	FR04014	no2	26.5
3	2019-06-20 21:00:00+00:00	FR04014	no2	24.9
4	2019-06-20 20:00:00+00:00	FR04014	no2	21.4

```
In [111]: air_quality_pm25 = pd.read_csv("air_quality_pm25_long.csv",parse_dates=True)
air_quality_pm25 = air_quality_pm25[["date.utc", "location","parameter", "value"]
air_quality_pm25.head()
```

Out[111]:

	date.utc	location	parameter	value
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

In [112]:

```
air_quality = pd.concat([air_quality_pm25, air_quality_no2], axis=0)
air_quality.head()
```

Out[112]:

	date.utc	location	parameter	value
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

In [113]:

```
print('Shape of the ``air_quality_pm25`` table: ', air_quality_pm25.shape)
print('Shape of the ``air_quality_no2`` table: ', air_quality_no2.shape)
print('Shape of the resulting ``air_quality`` table: ', air_quality.shape)
```

```
Shape of the ``air_quality_pm25`` table: (1110, 4)
Shape of the ``air_quality_no2`` table: (2068, 4)
Shape of the resulting ``air_quality`` table: (3178, 4)
```

In [114]:

```
air_quality = air_quality.sort_values("date.utc")
air_quality.head()
```

Out[114]:

	date.utc	location	parameter	value
2067	2019-05-07 01:00:00+00:00	London Westminster	no2	23.0
1003	2019-05-07 01:00:00+00:00	FR04014	no2	25.0
100	2019-05-07 01:00:00+00:00	BETR801	pm25	12.5
1098	2019-05-07 01:00:00+00:00	BETR801	no2	50.5
1109	2019-05-07 01:00:00+00:00	London Westminster	pm25	8.0

In [115]:

```
air_quality_ = pd.concat([air_quality_pm25, air_quality_no2], keys=["PM25", "NO2"])
air_quality_.head()
```

Out[115]:

	date.utc	location	parameter	value
0	2019-06-18 06:00:00+00:00	BETR801	pm25	18.0
1	2019-06-17 08:00:00+00:00	BETR801	pm25	6.5
PM25 2	2019-06-17 07:00:00+00:00	BETR801	pm25	18.5
3	2019-06-17 06:00:00+00:00	BETR801	pm25	16.0
4	2019-06-17 05:00:00+00:00	BETR801	pm25	7.5

```
In [116]: stations_coord = pd.read_csv("air_quality_stations.csv")
stations_coord.head()
```

```
Out[116]:
```

	location	coordinates.latitude	coordinates.longitude
0	BELAL01	51.23619	4.38522
1	BELHB23	51.17030	4.34100
2	BELLD01	51.10998	5.00486
3	BELLD02	51.12038	5.02155
4	BELR833	51.32766	4.36226

```
In [117]: air_quality = pd.merge(air_quality, stations_coord, how="left", on="location")
air_quality.head()
```

```
Out[117]:
```

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude
0	2019-05-07 01:00:00+00:00	London Westminster	no2	23.0	51.49467	-0.13193
1	2019-05-07 01:00:00+00:00	FR04014	no2	25.0	48.83724	2.39390
2	2019-05-07 01:00:00+00:00	FR04014	no2	25.0	48.83722	2.39390
3	2019-05-07 01:00:00+00:00	BETR801	pm25	12.5	51.20966	4.43182
4	2019-05-07 01:00:00+00:00	BETR801	no2	50.5	51.20966	4.43182

```
In [118]: air_quality_parameters = pd.read_csv("air_quality_parameters.csv")
air_quality_parameters.head()
```

```
Out[118]:
```

	id	description	name
0	bc	Black Carbon	BC
1	co	Carbon Monoxide	CO
2	no2	Nitrogen Dioxide	NO2
3	o3	Ozone	O3
4	pm10	Particulate matter less than 10 micrometers in...	PM10

```
In [119]: air_quality = pd.merge(air_quality, air_quality_parameters, how='left', left_on="id", right_on="parameter")
air_quality.head()
```

```
Out[119]:
```

	date.utc	location	parameter	value	coordinates.latitude	coordinates.longitude	id
0	2019-05-07 01:00:00+00:00	London Westminster	no2	23.0	51.49467	-0.13193	no2
1	2019-05-07 01:00:00+00:00	FR04014	no2	25.0	48.83724	2.39390	no2
2	2019-05-07 01:00:00+00:00	FR04014	no2	25.0	48.83722	2.39390	no2
3	2019-05-07 01:00:00+00:00	BETR801	pm25	12.5	51.20966	4.43182	pm25
4	2019-05-07 01:00:00+00:00	BETR801	no2	50.5	51.20966	4.43182	no2

```
In [120]: air_quality = pd.read_csv("air_quality_no2_long.csv")
air_quality = air_quality.rename(columns={"date.utc": "datetime"})
air_quality.head()
```

```
Out[120]:
```

	city	country	datetime	location	parameter	value	unit
0	Paris	FR	2019-06-21 00:00:00+00:00	FR04014	no2	20.0	µg/m³
1	Paris	FR	2019-06-20 23:00:00+00:00	FR04014	no2	21.8	µg/m³
2	Paris	FR	2019-06-20 22:00:00+00:00	FR04014	no2	26.5	µg/m³
3	Paris	FR	2019-06-20 21:00:00+00:00	FR04014	no2	24.9	µg/m³
4	Paris	FR	2019-06-20 20:00:00+00:00	FR04014	no2	21.4	µg/m³

```
In [121]: air_quality.city.unique()
```

```
Out[121]: array(['Paris', 'Antwerpen', 'London'], dtype=object)
```

```
In [122]: air_quality["datetime"] = pd.to_datetime(air_quality["datetime"])
air_quality["datetime"]
```

```
Out[122]: 0      2019-06-21 00:00:00+00:00
1      2019-06-20 23:00:00+00:00
2      2019-06-20 22:00:00+00:00
3      2019-06-20 21:00:00+00:00
4      2019-06-20 20:00:00+00:00
...
2063    2019-05-07 06:00:00+00:00
2064    2019-05-07 04:00:00+00:00
2065    2019-05-07 03:00:00+00:00
2066    2019-05-07 02:00:00+00:00
2067    2019-05-07 01:00:00+00:00
Name: datetime, Length: 2068, dtype: datetime64[ns, UTC]
```

```
In [123]: pd.read_csv("air_quality_no2_long.csv") #parse_dates=["datetime"]
```

Out[123]:

	city	country		date.utc	location	parameter	value	unit
0	Paris	FR	2019-06-21 00:00:00+00:00		FR04014	no2	20.0	µg/m³
1	Paris	FR	2019-06-20 23:00:00+00:00		FR04014	no2	21.8	µg/m³
2	Paris	FR	2019-06-20 22:00:00+00:00		FR04014	no2	26.5	µg/m³
3	Paris	FR	2019-06-20 21:00:00+00:00		FR04014	no2	24.9	µg/m³
4	Paris	FR	2019-06-20 20:00:00+00:00		FR04014	no2	21.4	µg/m³
...
2063	London	GB	2019-05-07 06:00:00+00:00	London Westminster		no2	26.0	µg/m³
2064	London	GB	2019-05-07 04:00:00+00:00	London Westminster		no2	16.0	µg/m³
2065	London	GB	2019-05-07 03:00:00+00:00	London Westminster		no2	19.0	µg/m³
2066	London	GB	2019-05-07 02:00:00+00:00	London Westminster		no2	19.0	µg/m³
2067	London	GB	2019-05-07 01:00:00+00:00	London Westminster		no2	23.0	µg/m³

2068 rows × 7 columns

```
In [124]: air_quality["datetime"].min(), air_quality["datetime"].max()
```

```
Out[124]: (Timestamp('2019-05-07 01:00:00+0000', tz='UTC'),
Timestamp('2019-06-21 00:00:00+0000', tz='UTC'))
```

```
In [125]: air_quality["datetime"].max() - air_quality["datetime"].min()
```

```
Out[125]: Timedelta('44 days 23:00:00')
```

```
In [126]: air_quality["month"] = air_quality["datetime"].dt.month
air_quality.head()
```

Out[126]:

	city	country	datetime	location	parameter	value	unit	month
0	Paris	FR	2019-06-21 00:00:00+00:00	FR04014	no2	20.0	µg/m³	6
1	Paris	FR	2019-06-20 23:00:00+00:00	FR04014	no2	21.8	µg/m³	6
2	Paris	FR	2019-06-20 22:00:00+00:00	FR04014	no2	26.5	µg/m³	6
3	Paris	FR	2019-06-20 21:00:00+00:00	FR04014	no2	24.9	µg/m³	6
4	Paris	FR	2019-06-20 20:00:00+00:00	FR04014	no2	21.4	µg/m³	6

```
In [127]: air_quality.groupby([air_quality["datetime"].dt.weekday, "location"])["value"]
```

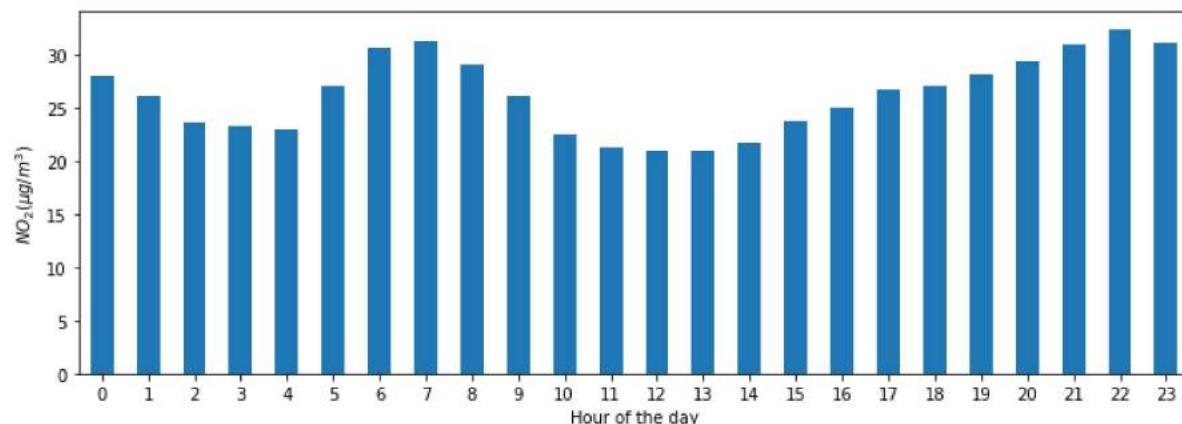
```
Out[127]: datetime  location
0           BETR801      27.875000
             FR04014      24.856250
             London Westminster  23.969697
1           BETR801      22.214286
             FR04014      30.999359
             London Westminster  24.885714
2           BETR801      21.125000
             FR04014      29.165753
             London Westminster  23.460432
3           BETR801      27.500000
             FR04014      28.600690
             London Westminster  24.780142
4           BETR801      28.400000
             FR04014      31.617986
             London Westminster  26.446809
5           BETR801      33.500000
             FR04014      25.266154
             London Westminster  24.977612
6           BETR801      21.896552
             FR04014      23.274306
             London Westminster  24.859155
Name: value, dtype: float64
```

```
In [128]: fig, axs = plt.subplots(figsize=(12, 4))
air_quality.groupby(air_quality["datetime"].dt.hour)[ "value"].mean().plot(kind="bar")
plt.xlabel("Hour of the day"); # custom x label using matplotlib
plt.ylabel("$NO_2 (\mu g/m^3)$");
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```

```
<IPython.core.display.Javascript object>
```



```
In [129]: no_2 = air_quality.pivot(index="datetime", columns="location", values="value")
no_2.head()
```

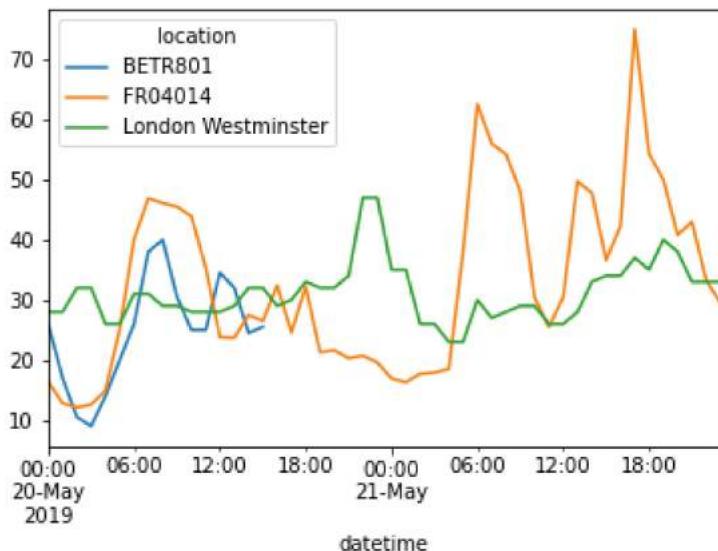
```
Out[129]:
```

location	BETR801	FR04014	London Westminster
datetime			
2019-05-07 01:00:00+00:00	50.5	25.0	23.0
2019-05-07 02:00:00+00:00	45.0	27.7	19.0
2019-05-07 03:00:00+00:00	NaN	50.4	19.0
2019-05-07 04:00:00+00:00	NaN	61.9	16.0
2019-05-07 05:00:00+00:00	NaN	72.4	NaN

```
In [130]: no_2.index.year, no_2.index.weekday
```

```
Out[130]: (Int64Index([2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019,
...
2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019, 2019],  
dtype='int64', name='datetime', length=1033),  
Int64Index([1, 1, 1, 1, 1, 1, 1, 1, 1, 1,dtype='int64', name='datetime', length=1033))
```

```
In [131]: no_2["2019-05-20":"2019-05-21"].plot();
```



```
In [132]: monthly_max = no_2.resample("M").max()
monthly_max
```

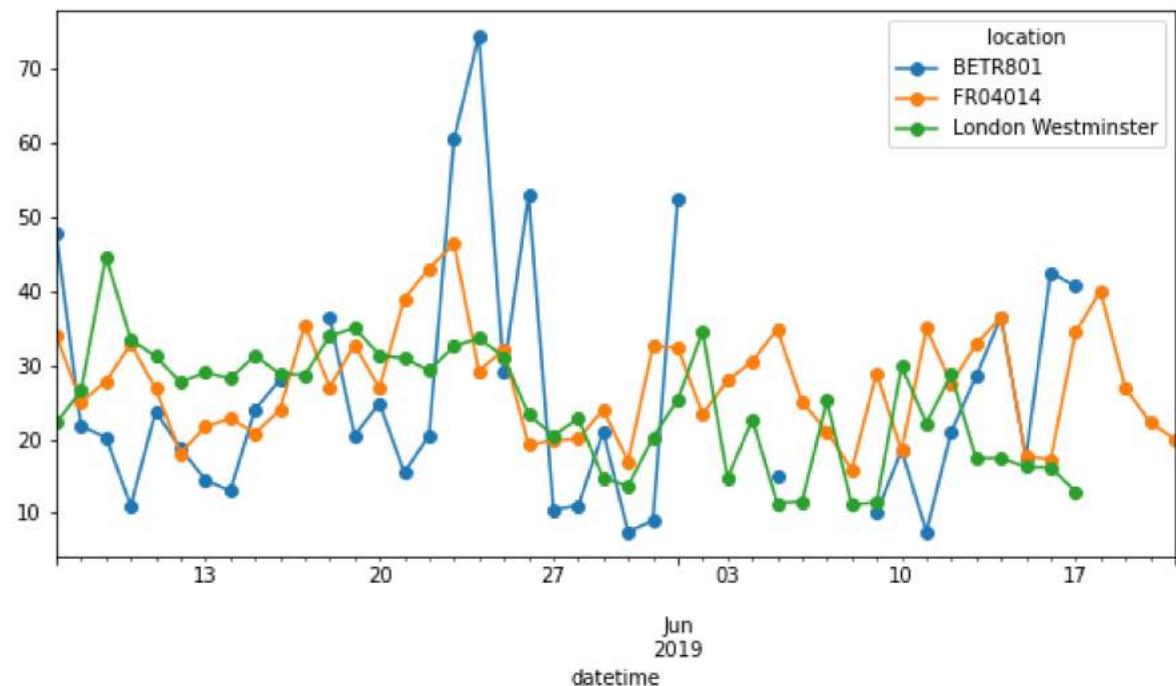
```
Out[132]:
```

location	BETR801	FR04014	London Westminster
datetime			
2019-05-31 00:00:00+00:00	74.5	97.0	97.0
2019-06-30 00:00:00+00:00	52.5	84.7	52.0

```
In [133]: monthly_max.index.freq
```

```
Out[133]: <MonthEnd>
```

```
In [134]: no_2.resample("D").mean().plot(style="-o", figsize=(10, 5));
```



```
In [1]: import pandas as pd  
import numpy as np
```

```
In [4]: url = ("https://raw.github.com/pandas-dev/pandas/main/pandas/tests/io/data/csv/tips.csv")  
tips = pd.read_csv(url)  
tips.head()
```

Out[4]:

	total_bill	tip	sex	smoker	day	time	size	
0	16.99	1.01	Female		No	Sun	Dinner	2
1	10.34	1.66	Male		No	Sun	Dinner	3
2	21.01	3.50	Male		No	Sun	Dinner	3
3	23.68	3.31	Male		No	Sun	Dinner	2
4	24.59	3.61	Female		No	Sun	Dinner	4

```
In [3]: tips[["total_bill", "tip", "smoker", "time"]]
```

Out[3]:

	total_bill	tip	smoker	time
0	16.99	1.01	No	Dinner
1	10.34	1.66	No	Dinner
2	21.01	3.50	No	Dinner
3	23.68	3.31	No	Dinner
4	24.59	3.61	No	Dinner
...
239	29.03	5.92	No	Dinner
240	27.18	2.00	Yes	Dinner
241	22.67	2.00	Yes	Dinner
242	17.82	1.75	No	Dinner
243	18.78	3.00	No	Dinner

244 rows × 4 columns

```
In [4]: tips.assign(tip_rate=tips["tip"] / tips["total_bill"])
```

```
Out[4]:
```

	total_bill	tip	sex	smoker	day	time	size	tip_rate
0	16.99	1.01	Female	No	Sun	Dinner	2	0.059447
1	10.34	1.66	Male	No	Sun	Dinner	3	0.160542
2	21.01	3.50	Male	No	Sun	Dinner	3	0.166587
3	23.68	3.31	Male	No	Sun	Dinner	2	0.139780
4	24.59	3.61	Female	No	Sun	Dinner	4	0.146808
...
239	29.03	5.92	Male	No	Sat	Dinner	3	0.203927
240	27.18	2.00	Female	Yes	Sat	Dinner	2	0.073584
241	22.67	2.00	Male	Yes	Sat	Dinner	2	0.088222
242	17.82	1.75	Male	No	Sat	Dinner	2	0.098204
243	18.78	3.00	Female	No	Thur	Dinner	2	0.159744

244 rows × 8 columns

```
In [5]: is_dinner = tips["time"] == "Dinner"  
is_dinner
```

```
Out[5]: 0      True  
1      True  
2      True  
3      True  
4      True  
...  
239    True  
240    True  
241    True  
242    True  
243    True  
Name: time, Length: 244, dtype: bool
```

```
In [6]: is_dinner.value_counts()
```

```
Out[6]: True    176  
False   68  
Name: time, dtype: int64
```

```
In [7]: tips[is_dinner]
```

```
Out[7]:
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

176 rows × 7 columns

```
In [8]: tips[(tips["time"] == "Dinner") & (tips["tip"] > 5.00)]
```

```
Out[8]:
```

	total_bill	tip	sex	smoker	day	time	size
23	39.42	7.58	Male	No	Sat	Dinner	4
44	30.40	5.60	Male	No	Sun	Dinner	4
47	32.40	6.00	Male	No	Sun	Dinner	4
52	34.81	5.20	Female	No	Sun	Dinner	4
59	48.27	6.73	Male	No	Sat	Dinner	4
116	29.93	5.07	Male	No	Sun	Dinner	4
155	29.85	5.14	Female	No	Sun	Dinner	5
170	50.81	10.00	Male	Yes	Sat	Dinner	3
172	7.25	5.15	Male	Yes	Sun	Dinner	2
181	23.33	5.65	Male	Yes	Sun	Dinner	2
183	23.17	6.50	Male	Yes	Sun	Dinner	4
211	25.89	5.16	Male	Yes	Sat	Dinner	4
212	48.33	9.00	Male	No	Sat	Dinner	4
214	28.17	6.50	Female	Yes	Sat	Dinner	3
239	29.03	5.92	Male	No	Sat	Dinner	3

```
In [9]: tips[(tips["size"] >= 5) | (tips["total_bill"] > 45)]
```

```
Out[9]:
```

	total_bill	tip	sex	smoker	day	time	size
59	48.27	6.73	Male	No	Sat	Dinner	4
125	29.80	4.20	Female	No	Thur	Lunch	6
141	34.30	6.70	Male	No	Thur	Lunch	6
142	41.19	5.00	Male	No	Thur	Lunch	5
143	27.05	5.00	Female	No	Thur	Lunch	6
155	29.85	5.14	Female	No	Sun	Dinner	5
156	48.17	5.00	Male	No	Sun	Dinner	6
170	50.81	10.00	Male	Yes	Sat	Dinner	3
182	45.35	3.50	Male	Yes	Sun	Dinner	3
185	20.69	5.00	Male	No	Sun	Dinner	5
187	30.46	2.00	Male	Yes	Sun	Dinner	5
212	48.33	9.00	Male	No	Sat	Dinner	4
216	28.15	3.00	Male	Yes	Sat	Dinner	5

```
In [10]: tips.groupby("sex").size()
```

```
Out[10]: sex
Female    87
Male     157
dtype: int64
```

```
In [11]: tips.groupby("sex").count()
```

```
Out[11]:
```

	total_bill	tip	smoker	day	time	size
Female	87	87	87	87	87	87
Male	157	157	157	157	157	157

```
In [12]: tips.groupby("sex")["total_bill"].count()
```

```
Out[12]: sex
Female    87
Male     157
Name: total_bill, dtype: int64
```

```
In [13]: tips.groupby("day").agg({"tip": np.mean, "day": np.size})
```

```
Out[13]:
```

day	tip	day
Fri	2.734737	19
Sat	2.993103	87
Sun	3.255132	76
Thur	2.771452	62

```
In [14]: tips.groupby(["smoker", "day"]).agg({"tip": [np.size, np.mean]})
```

```
Out[14]:
```

		tip	
		size	mean
smoker	day		
No	Fri	4	2.812500
	Sat	45	3.102889
	Sun	57	3.167895
	Thur	45	2.673778
Yes	Fri	15	2.714000
	Sat	42	2.875476
	Sun	19	3.516842
	Thur	17	3.030000

```
In [15]: tips.nlargest(10 + 5, columns="tip").tail(2)
```

```
Out[15]:
```

	total_bill	tip	sex	smoker	day	time	size
85	34.83	5.17	Female	No	Thur	Lunch	4
211	25.89	5.16	Male	Yes	Sat	Dinner	4

In [17]:

```
(  
    tips.assign(  
        rn=tips.sort_values(["total_bill"], ascending=False)  
        .groupby(["day"])  
        .cumcount()  
        + 1  
    )  
    .query("rn < 3")  
    .sort_values(["day", "rn"])  
)
```

Out[17]:

	total_bill	tip	sex	smoker	day	time	size	rn
95	40.17	4.73	Male	Yes	Fri	Dinner	4	1
90	28.97	3.00	Male	Yes	Fri	Dinner	2	2
170	50.81	10.00	Male	Yes	Sat	Dinner	3	1
212	48.33	9.00	Male	No	Sat	Dinner	4	2
156	48.17	5.00	Male	No	Sun	Dinner	6	1
182	45.35	3.50	Male	Yes	Sun	Dinner	3	2
197	43.11	5.00	Female	Yes	Thur	Lunch	4	1
142	41.19	5.00	Male	No	Thur	Lunch	5	2

In [18]:

```
(  
    tips.assign(  
        rnk=tips.groupby(["day"])["total_bill"].rank(  
            method="first", ascending=False  
        )  
    )  
    .query("rnk < 3")  
    .sort_values(["day", "rnk"])  
)
```

Out[18]:

	total_bill	tip	sex	smoker	day	time	size	rnk
95	40.17	4.73	Male	Yes	Fri	Dinner	4	1.0
90	28.97	3.00	Male	Yes	Fri	Dinner	2	2.0
170	50.81	10.00	Male	Yes	Sat	Dinner	3	1.0
212	48.33	9.00	Male	No	Sat	Dinner	4	2.0
156	48.17	5.00	Male	No	Sun	Dinner	6	1.0
182	45.35	3.50	Male	Yes	Sun	Dinner	3	2.0
197	43.11	5.00	Female	Yes	Thur	Lunch	4	1.0
142	41.19	5.00	Male	No	Thur	Lunch	5	2.0

In [19]:

```
(  
    tips[tips["tip"] < 2]  
    .assign(rnk_min=tips.groupby(["sex"])["tip"].rank(method="min"))  
    .query("rnk_min < 3")  
    .sort_values(["sex", "rnk_min"])  
)
```

Out[19]:

	total_bill	tip	sex	smoker	day	time	size	rnk_min
67	3.07	1.00	Female	Yes	Sat	Dinner	1	1.0
92	5.75	1.00	Female	Yes	Fri	Dinner	2	1.0
111	7.25	1.00	Female	No	Sat	Dinner	1	1.0
236	12.60	1.00	Male	Yes	Sat	Dinner	2	1.0
237	32.83	1.17	Male	Yes	Sat	Dinner	2	2.0

In [20]: tips.loc[tips["tip"] < 2, "tip"] *= 2

In [21]: tips = tips.loc[tips["tip"] <= 9]

In [23]: tips = pd.read_csv("tips.csv", sep="\t", header=None)
alternatively, read_table is an alias to read_csv with tab delimiter
tips = pd.read_table("tips.csv", header=None)

In [24]: tips.to_excel("./tips.xlsx")

In [25]: tips_df = pd.read_excel("./tips.xlsx", index_col=0)

In [26]: tips.head(5)

Out[26]:

	0
0	total_bill,tip,sex,smoker,day,time,size
1	16.99,1.01,Female,No,Sun,Dinner,2
2	10.34,1.66,Male,No,Sun,Dinner,3
3	21.01,3.5,Male,No,Sun,Dinner,3
4	23.68,3.31,Male,No,Sun,Dinner,2

```
In [27]: tips = pd.read_csv("tips.csv", sep="\t", header=None)
# alternatively, read_table is an alias to read_csv with tab delimiter
tips = pd.read_table("tips.csv", header=None)
tips.head()
```

```
Out[27]:
```

0	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
In [28]: url = ("https://raw.github.com/pandas-dev/pandas/main/pandas/tests/io/data/csv/tips.csv")
tips = pd.read_csv(url)
tips.head()
```

```
Out[28]:
```

0	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
In [29]: tips["total_bill"] = tips["total_bill"] - 2
tips["new_bill"] = tips["total_bill"] / 2
tips.head()
```

```
Out[29]:
```

0	total_bill	tip	sex	smoker	day	time	size	new_bill
0	14.99	1.01	Female	No	Sun	Dinner	2	7.495
1	8.34	1.66	Male	No	Sun	Dinner	3	4.170
2	19.01	3.50	Male	No	Sun	Dinner	3	9.505
3	21.68	3.31	Male	No	Sun	Dinner	2	10.840
4	22.59	3.61	Female	No	Sun	Dinner	4	11.295

```
In [30]: tips = tips.drop("new_bill", axis=1)
```

```
In [31]: tips[tips["total_bill"] > 10]
```

Out[31]:

	total_bill	tip	sex	smoker	day	time	size
0	14.99	1.01	Female	No	Sun	Dinner	2
2	19.01	3.50	Male	No	Sun	Dinner	3
3	21.68	3.31	Male	No	Sun	Dinner	2
4	22.59	3.61	Female	No	Sun	Dinner	4
5	23.29	4.71	Male	No	Sun	Dinner	4
...
239	27.03	5.92	Male	No	Sat	Dinner	3
240	25.18	2.00	Female	Yes	Sat	Dinner	2
241	20.67	2.00	Male	Yes	Sat	Dinner	2
242	15.82	1.75	Male	No	Sat	Dinner	2
243	16.78	3.00	Female	No	Thur	Dinner	2

204 rows × 7 columns

```
In [32]: tips["bucket"] = np.where(tips["total_bill"] < 10, "low", "high")  
tips
```

Out[32]:

	total_bill	tip	sex	smoker	day	time	size	bucket
0	14.99	1.01	Female	No	Sun	Dinner	2	high
1	8.34	1.66	Male	No	Sun	Dinner	3	low
2	19.01	3.50	Male	No	Sun	Dinner	3	high
3	21.68	3.31	Male	No	Sun	Dinner	2	high
4	22.59	3.61	Female	No	Sun	Dinner	4	high
...
239	27.03	5.92	Male	No	Sat	Dinner	3	high
240	25.18	2.00	Female	Yes	Sat	Dinner	2	high
241	20.67	2.00	Male	Yes	Sat	Dinner	2	high
242	15.82	1.75	Male	No	Sat	Dinner	2	high
243	16.78	3.00	Female	No	Thur	Dinner	2	high

244 rows × 8 columns

```
In [33]: tips["date1"] = pd.Timestamp("2013-01-15")
tips["date2"] = pd.Timestamp("2015-02-15")
tips["date1_year"] = tips["date1"].dt.year
tips["date2_month"] = tips["date2"].dt.month
tips["date1_next"] = tips["date1"] + pd.offsets.MonthBegin()
tips["months_between"] = tips["date2"].dt.to_period("M") - tips["date1"].dt.to_
tips[["date1", "date2", "date1_year", "date2_month", "date1_next", "months_betw
```

Out[33]:

	date1	date2	date1_year	date2_month	date1_next	months_between
0	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
1	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
2	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
3	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
4	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
...
239	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
240	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
241	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
242	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>
243	2013-01-15	2015-02-15	2013		2 2013-02-01	<25 * MonthEnds>

244 rows × 6 columns

```
In [34]: tips[["sex", "total_bill", "tip"]]
```

Out[34]:

	sex	total_bill	tip
0	Female	14.99	1.01
1	Male	8.34	1.66
2	Male	19.01	3.50
3	Male	21.68	3.31
4	Female	22.59	3.61
...
239	Male	27.03	5.92
240	Female	25.18	2.00
241	Male	20.67	2.00
242	Male	15.82	1.75
243	Female	16.78	3.00

244 rows × 3 columns

```
In [35]: tips.drop("sex", axis=1)
```

Out[35]:

	total_bill	tip	smoker	day	time	size	bucket	date1	date2	date1_year	date2_month
0	14.99	1.01	No	Sun	Dinner	2	high	2013-01-15	2015-02-15	2013	2
1	8.34	1.66	No	Sun	Dinner	3	low	2013-01-15	2015-02-15	2013	2
2	19.01	3.50	No	Sun	Dinner	3	high	2013-01-15	2015-02-15	2013	2
3	21.68	3.31	No	Sun	Dinner	2	high	2013-01-15	2015-02-15	2013	2
4	22.59	3.61	No	Sun	Dinner	4	high	2013-01-15	2015-02-15	2013	2
...
239	27.03	5.92	No	Sat	Dinner	3	high	2013-01-15	2015-02-15	2013	2
240	25.18	2.00	Yes	Sat	Dinner	2	high	2013-01-15	2015-02-15	2013	2
241	20.67	2.00	Yes	Sat	Dinner	2	high	2013-01-15	2015-02-15	2013	2
242	15.82	1.75	No	Sat	Dinner	2	high	2013-01-15	2015-02-15	2013	2
243	16.78	3.00	No	Thur	Dinner	2	high	2013-01-15	2015-02-15	2013	2

244 rows × 13 columns



```
In [40]: tips.rename(columns={"total_bill": "total_bill_2"})
```

Out[40]:

	total_bill_2	tip	sex	smoker	day	time	size	bucket	date1	date2	date1_year	da
67	1.07	1.00	Female	Yes	Sat	Dinner	1	low	2013-01-15	2015-02-15	2013	
92	3.75	1.00	Female	Yes	Fri	Dinner	2	low	2013-01-15	2015-02-15	2013	
111	5.25	1.00	Female	No	Sat	Dinner	1	low	2013-01-15	2015-02-15	2013	
145	6.35	1.50	Female	No	Thur	Lunch	2	low	2013-01-15	2015-02-15	2013	
135	6.51	1.25	Female	No	Thur	Lunch	2	low	2013-01-15	2015-02-15	2013	
...
182	43.35	3.50	Male	Yes	Sun	Dinner	3	high	2013-01-15	2015-02-15	2013	
156	46.17	5.00	Male	No	Sun	Dinner	6	high	2013-01-15	2015-02-15	2013	
59	46.27	6.73	Male	No	Sat	Dinner	4	high	2013-01-15	2015-02-15	2013	
212	46.33	9.00	Male	No	Sat	Dinner	4	high	2013-01-15	2015-02-15	2013	
170	48.81	10.00	Male	Yes	Sat	Dinner	3	high	2013-01-15	2015-02-15	2013	

244 rows × 14 columns

```
In [38]: tips = tips.sort_values(["sex", "total_bill"])
tips.head(2)
```

Out[38]:

	total_bill	tip	sex	smoker	day	time	size	bucket	date1	date2	date1_year	date2_m
67	1.07	1.0	Female	Yes	Sat	Dinner	1	low	2013-01-15	2015-02-15	2013	
92	3.75	1.0	Female	Yes	Fri	Dinner	2	low	2013-01-15	2015-02-15	2013	

```
In [41]: tips["time"].str.len()
```

```
Out[41]: 67      6  
92      6  
111     6  
145     5  
135     5  
..  
182      6  
156      6  
59       6  
212      6  
170      6  
Name: time, Length: 244, dtype: int64
```

```
In [42]: tips["time"].str.rstrip().str.len()
```

```
Out[42]: 67      6  
92      6  
111     6  
145     5  
135     5  
..  
182      6  
156      6  
59       6  
212      6  
170      6  
Name: time, Length: 244, dtype: int64
```

```
In [43]: tips["sex"].str.find("ale")
```

```
Out[43]: 67      3  
92      3  
111     3  
145     3  
135     3  
..  
182      1  
156      1  
59       1  
212      1  
170      1  
Name: sex, Length: 244, dtype: int64
```

```
In [44]: tips["sex"].str[0:1]
```

```
Out[44]: 67      F  
92      F  
111     F  
145     F  
135     F  
..  
182      M  
156      M  
59       M  
212      M  
170      M  
Name: sex, Length: 244, dtype: object
```

```
In [45]: pd.pivot_table(tips, values="tip", index=["size"], columns=["sex"], aggfunc=np.
```

```
Out[45]:   sex    Female      Male  
size  
---  
1  1.276667  1.920000  
2  2.528448  2.614184  
3  3.250000  3.476667  
4  4.021111  4.172143  
5  5.140000  3.750000  
6  4.600000  5.850000
```

```
In [54]: tips.iloc[1:2,0:3]
```

```
Out[54]:    total_bill  tip      sex  
92        3.75  1.0  Female
```

```
In [56]: tips == "3.75"
```

```
Out[56]:
```

	total_bill	tip	sex	smoker	day	time	size	bucket	date1	date2	date1_year	date2_
67	False	False	False	False	False	False	False	False	False	False	False	False
92	False	False	False	False	False	False	False	False	False	False	False	False
111	False	False	False	False	False	False	False	False	False	False	False	False
145	False	False	False	False	False	False	False	False	False	False	False	False
135	False	False	False	False	False	False	False	False	False	False	False	False
...
182	False	False	False	False	False	False	False	False	False	False	False	False
156	False	False	False	False	False	False	False	False	False	False	False	False
59	False	False	False	False	False	False	False	False	False	False	False	False
212	False	False	False	False	False	False	False	False	False	False	False	False
170	False	False	False	False	False	False	False	False	False	False	False	False

244 rows × 14 columns



```
In [57]: tips["day"].str.contains("S")
```

```
Out[57]:
```

67	True
92	False
111	True
145	False
135	False
...	
182	True
156	True
59	True
212	True
170	True

Name: day, Length: 244, dtype: bool

```
In [58]: tips.replace("Thu", "Thursday")
```

Out[58]:

	total_bill	tip	sex	smoker	day	time	size	bucket	date1	date2	date1_year	date
67	1.07	1.00	Female	Yes	Sat	Dinner	1	low	2013-01-15	2015-02-15	2013	
92	3.75	1.00	Female	Yes	Fri	Dinner	2	low	2013-01-15	2015-02-15	2013	
111	5.25	1.00	Female	No	Sat	Dinner	1	low	2013-01-15	2015-02-15	2013	
145	6.35	1.50	Female	No	Thur	Lunch	2	low	2013-01-15	2015-02-15	2013	
135	6.51	1.25	Female	No	Thur	Lunch	2	low	2013-01-15	2015-02-15	2013	
...
182	43.35	3.50	Male	Yes	Sun	Dinner	3	high	2013-01-15	2015-02-15	2013	
156	46.17	5.00	Male	No	Sun	Dinner	6	high	2013-01-15	2015-02-15	2013	
59	46.27	6.73	Male	No	Sat	Dinner	4	high	2013-01-15	2015-02-15	2013	
212	46.33	9.00	Male	No	Sat	Dinner	4	high	2013-01-15	2015-02-15	2013	
170	48.81	10.00	Male	Yes	Sat	Dinner	3	high	2013-01-15	2015-02-15	2013	

244 rows × 14 columns



```
In [59]: tips_summed = tips.groupby(["sex", "smoker"])[["total_bill", "tip"]].sum()  
tips_summed
```

Out[59]:

		total_bill	tip
	sex	smoker	
Female		No	869.68 149.77
		Yes	527.27 96.74
Male		No	1725.75 302.00
		Yes	1217.07 183.07

```
In [61]: gb = tips.groupby("smoker")["total_bill"]
tips["adj_total_bill"] = tips["total_bill"] - gb.transform("mean")
tips.head(2)
```

```
Out[61]:
```

	total_bill	tip	sex	smoker	day	time	size	bucket	date1	date2	date1_year	date2_mon
67	1.07	1.0	Female	Yes	Sat	Dinner	1	low	2013-01-15	2015-02-15		2013
92	3.75	1.0	Female	Yes	Fri	Dinner	2	low	2013-01-15	2015-02-15		2013

```
In [62]: tips.groupby(["sex", "smoker"]).first()
```

```
Out[62]:
```

	total_bill	tip	day	time	size	bucket	date1	date2	date1_year	date2_mon	
sex	smoker										
Female	No	5.25	1.00	Sat	Dinner	1	low	2013-01-15	2015-02-15		2013
	Yes	1.07	1.00	Sat	Dinner	1	low	2013-01-15	2015-02-15		2013
Male	No	5.51	2.00	Thur	Lunch	2	low	2013-01-15	2015-02-15		2013
	Yes	5.25	5.15	Sun	Dinner	2	low	2013-01-15	2015-02-15		2013

```
In [64]: url = ("https://raw.github.com/pandas-dev/pandas/main/pandas/tests/io/data/csv/tips.csv")
tips = pd.read_csv(url)
tips.head(2)
```

```
Out[64]:
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3

In [65]:

```
tips[tips["total_bill"] > 10]
```

Out[65]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
...
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2

227 rows × 7 columns

In [66]:

```
tips[["sex", "total_bill", "tip"]]
```

Out[66]:

	sex	total_bill	tip
0	Female	16.99	1.01
1	Male	10.34	1.66
2	Male	21.01	3.50
3	Male	23.68	3.31
4	Female	24.59	3.61
...
239	Male	29.03	5.92
240	Female	27.18	2.00
241	Male	22.67	2.00
242	Male	17.82	1.75
243	Female	18.78	3.00

244 rows × 3 columns

```
In [67]: tips = tips.sort_values(["sex", "total_bill"])
tips
```

```
Out[67]:
```

	total_bill	tip	sex	smoker	day	time	size
67	3.07	1.00	Female	Yes	Sat	Dinner	1
92	5.75	1.00	Female	Yes	Fri	Dinner	2
111	7.25	1.00	Female	No	Sat	Dinner	1
145	8.35	1.50	Female	No	Thur	Lunch	2
135	8.51	1.25	Female	No	Thur	Lunch	2
...
182	45.35	3.50	Male	Yes	Sun	Dinner	3
156	48.17	5.00	Male	No	Sun	Dinner	6
59	48.27	6.73	Male	No	Sat	Dinner	4
212	48.33	9.00	Male	No	Sat	Dinner	4
170	50.81	10.00	Male	Yes	Sat	Dinner	3

244 rows × 7 columns

```
In [5]: print(tips.iloc[-20:, :12].to_string())
```

	total_bill	tip	sex	smoker	day	time	size
224	13.42	1.58	Male	Yes	Fri	Lunch	2
225	16.27	2.50	Female	Yes	Fri	Lunch	2
226	10.09	2.00	Female	Yes	Fri	Lunch	2
227	20.45	3.00	Male	No	Sat	Dinner	4
228	13.28	2.72	Male	No	Sat	Dinner	2
229	22.12	2.88	Female	Yes	Sat	Dinner	2
230	24.01	2.00	Male	Yes	Sat	Dinner	4
231	15.69	3.00	Male	Yes	Sat	Dinner	3
232	11.61	3.39	Male	No	Sat	Dinner	2
233	10.77	1.47	Male	No	Sat	Dinner	2
234	15.53	3.00	Male	Yes	Sat	Dinner	2
235	10.07	1.25	Male	No	Sat	Dinner	2
236	12.60	1.00	Male	Yes	Sat	Dinner	2
237	32.83	1.17	Male	Yes	Sat	Dinner	2
238	35.83	4.67	Female	No	Sat	Dinner	3
239	29.03	5.92	Male	No	Sat	Dinner	3
240	27.18	2.00	Female	Yes	Sat	Dinner	2
241	22.67	2.00	Male	Yes	Sat	Dinner	2
242	17.82	1.75	Male	No	Sat	Dinner	2
243	18.78	3.00	Female	No	Thur	Dinner	2