In [1]:

```
#1
import pandas as pd
import numpy as np
```

In [2]:

```
#2
pd.__version__
```

Out[2]:

```
'1.3.4'
```

In [3]:

```
#3
pd.show_versions()
```

```
INSTALLED VERSIONS
------------------
commit           : 945c9ed766a61c7d2c0a7cbb251b6edebf9cb7d5
python           : 3.9.7.final.0
python-bits      : 64
OS               : Windows
OS-release       : 10
Version          : 10.0.19042
machine          : AMD64
processor        : Intel64 Family 6 Model 140 Stepping 1, GenuineIntel
byteorder        : little
LC_ALL           : None
LANG             : None
LOCALE           : English_India.1252

pandas           : 1.3.4
numpy            : 1.21.3
pytz             : 2021.3
dateutil         : 2.8.2
pip              : 21.3.1
setuptools       : 58.5.2
Cython           : None
pytest           : None
hypothesis       : None
sphinx           : None
blosc            : None
feather          : None
xlsxwriter       : None
lxml.etree       : None
html5lib         : None
pymysql          : None
psycopg2         : None
jinja2           : 3.0.2
IPython          : 7.29.0
pandas_datareader: None
bs4              : 4.10.0
bottleneck       : None
fsspec           : None
fastparquet      : None
gcsfs            : None
matplotlib       : 3.4.3
numexpr          : None
odfpy            : None
openpyxl         : None
pandas_gbq       : None
pyarrow          : None
pyxlsb           : None
s3fs             : None
scipy            : 1.7.1
sqlalchemy       : None
tables           : None
tabulate         : None
```

```
xarray           : None
xlrd             : None
xlwt             : None
numba            : None
```

In [4]:

```
#4
data = {
    'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'do
    'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
    'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']
}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
```

In [5]:

```
df = pd.DataFrame.from_dict(data)
labels = pd.DataFrame.from_dict(labels)
```

In [6]:

```
df['labels'] = labels
df = df.set_index('labels')

df
```

Out[6]:

	animal	age	visits	priority
labels				
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	2.0	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no

In [7]:

```
#5
df.describe()
```

Out[7]:

	age	visits
count	8.000000	10.000000
mean	3.437500	1.900000
std	2.007797	0.875595
min	0.500000	1.000000
25%	2.375000	1.000000
50%	3.000000	2.000000
75%	4.625000	2.750000
max	7.000000	3.000000

In [8]:

```
#6
df1 = df.head(3)
df1
```

Out[8]:

```
       animal  age  visits  priority
labels
     a    cat  2.5       1       yes
     b    cat  3.0       3       yes
     c  snake  0.5       2        no
```

In [9]:

```
#7
df[['animal', 'age']]
```

Out[9]:

```
       animal  age
labels
     a    cat  2.5
     b    cat  3.0
     c  snake  0.5
     d    dog  NaN
     e    dog  5.0
     f    cat  2.0
     g  snake  4.5
     h    cat  NaN
     i    dog  7.0
     j    dog  3.0
```

In [10]:

```
#8
df[['animal', 'age']].iloc[[3,4,8]]
```

Out[10]:

```
       animal  age
labels
     d    dog  NaN
     e    dog  5.0
     i    dog  7.0
```

In [11]:

```
#9
df[df['visits']>3]
```

Out[11]:

```
       animal  age  visits  priority
labels
```

In [12]:

```
#10
temp = df[df['age'].isnull()]
```

In [13]:
```
temp
```

Out[13]:

	animal	age	visits	priority
labels				
d	dog	NaN	3	yes
h	cat	NaN	1	yes

In [28]:
```
#11
df = pd.DataFrame(data , index=labels)

print(df[(df['animal'] == 'cat') & (df['age'] < 3)])
```

```
     animal  age  visits priority
(a,)    cat  2.5       1      yes
(f,)    cat  2.0       3       no
```

In [29]:
```
#12
df[(df['age'] <= 4) & (df['age'] >= 2)]
```

Out[29]:

	animal	age	visits	priority
(a,)	cat	2.5	1	yes
(b,)	cat	3.0	3	yes
(f,)	cat	2.0	3	no
(j,)	dog	3.0	1	no

In [37]:
```
#13
data = {
    'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'do
    'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
    'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']
}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df = pd.DataFrame(data , index=labels)
print("\nOriginal data frame:")
print(df)
print("\nChange the score in row 'd' to 11.5:")
df.loc['f', 'age'] = 1.5
print(df)
```

```
Original data frame:
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
d    dog  NaN       3      yes
e    dog  5.0       2       no
f    cat  2.0       3       no
g  snake  4.5       1       no
h    cat  NaN       1      yes
i    dog  7.0       2       no
j    dog  3.0       1       no

Change the score in row 'd' to 11.5:
  animal  age  visits priority
a    cat  2.5       1      yes
b    cat  3.0       3      yes
c  snake  0.5       2       no
```

```
d     dog  NaN        3      yes
e     dog  5.0        2       no
f     cat  1.5        3       no
g   snake  4.5        1       no
h     cat  NaN        1      yes
i     dog  7.0        2       no
j     dog  3.0        1       no
```

In [40]:

```
#14
data = {
    'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'do
    'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
    'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']
}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df['visits'].sum()
```

Out[40]:

```
19
```

In [41]:

```
#15
data = {
    'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'do
    'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
    'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']
}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df.groupby('animal')['age'].mean()
```

Out[41]:

```
animal
cat      2.333333
dog      5.000000
snake    2.500000
Name: age, dtype: float64
```

In [42]:

```
#16
df.loc['k'] = ['snake', 3.5, 3, 'yes']
df
```

Out[42]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	1.5	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no
k	snake	3.5	3	yes

In [43]:

```
df = df.drop('k')
```

```
df
```

Out[43]:

	animal	age	visits	priority
a	cat	2.5	1	yes
b	cat	3.0	3	yes
c	snake	0.5	2	no
d	dog	NaN	3	yes
e	dog	5.0	2	no
f	cat	1.5	3	no
g	snake	4.5	1	no
h	cat	NaN	1	yes
i	dog	7.0	2	no
j	dog	3.0	1	no

In [44]:

```
#17
df['animal'].value_counts()
```

Out[44]:

```
cat      4
dog      4
snake    2
Name: animal, dtype: int64
```

In [45]:

```
#18
df.sort_values('age')
```

Out[45]:

	animal	age	visits	priority
c	snake	0.5	2	no
f	cat	1.5	3	no
a	cat	2.5	1	yes
b	cat	3.0	3	yes
j	dog	3.0	1	no
g	snake	4.5	1	no
e	dog	5.0	2	no
i	dog	7.0	2	no
d	dog	NaN	3	yes
h	cat	NaN	1	yes

In [46]:

```
df.sort_values('visits')
```

Out[46]:

	animal	age	visits	priority
a	cat	2.5	1	yes
g	snake	4.5	1	no
h	cat	NaN	1	yes
j	dog	3.0	1	no
c	snake	0.5	2	no

e	dog	5.0	2	no
i	dog	7.0	2	no
b	cat	3.0	3	yes
d	dog	NaN	3	yes
f	cat	1.5	3	no

In [66]:

```
#18
df.sort_values(by=['age', 'visits'], ascending=[False, True])
```

Out[66]:

	animal	age	visits	priority
i	dog	7.0	2	NaN
e	dog	5.0	2	NaN
g	snake	4.5	1	NaN
j	dog	3.0	1	NaN
b	cat	3.0	3	NaN
a	cat	2.5	1	NaN
f	cat	1.5	3	NaN
c	snake	0.5	2	NaN
h	cat	NaN	1	NaN
d	dog	NaN	3	NaN

In [67]:

```
data = {
    'animal': ['cat', 'cat', 'snake', 'dog', 'dog', 'cat', 'snake', 'cat', 'dog', 'dc
    'age': [2.5, 3, 0.5, np.nan, 5, 2, 4.5, np.nan, 7, 3],
    'visits': [1, 3, 2, 3, 2, 3, 1, 1, 2, 1],
    'priority': ['yes', 'yes', 'no', 'yes', 'no', 'no', 'no', 'yes', 'no', 'no']
}

labels = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j']
df = pd.DataFrame.from_dict(data)
labels = pd.DataFrame.from_dict(labels)
df['labels'] = labels
df = df.set_index('labels')

df
df['priority'] = df['priority'].map(
                    {'yes':True ,'no':False})

# show the dataframe
df
```

Out[67]:

	animal	age	visits	priority
labels				
a	cat	2.5	1	True
b	cat	3.0	3	True
c	snake	0.5	2	False
d	dog	NaN	3	True
e	dog	5.0	2	False
f	cat	2.0	3	False

g	snake	4.5	1	False
h	cat	NaN	1	True
i	dog	7.0	2	False
j	dog	3.0	1	False

In [69]:

```
#20
df['animal'] = df['animal'].replace('snake', 'python')
df
```

Out[69]:

	animal	**age**	**visits**	**priority**
labels				
a	cat	2.5	1	True
b	cat	3.0	3	True
c	python	0.5	2	False
d	dog	NaN	3	True
e	dog	5.0	2	False
f	cat	2.0	3	False
g	python	4.5	1	False
h	cat	NaN	1	True
i	dog	7.0	2	False
j	dog	3.0	1	False

In [77]:

```
#21
df.pivot_table(index='animal', columns='visits', values='age', aggfunc='mean')
```

Out[77]:

visits	**1**	**2**	**3**
animal			
cat	2.5	NaN	2.5
dog	3.0	6.0	NaN
python	4.5	0.5	NaN

In [78]:

```
#22
df = pd.DataFrame({'A': [1, 2, 2, 3, 4, 5, 5, 5, 6, 7, 7]})
df.loc[df['A'].shift() != df['A']]
```

Out[78]:

	A
0	1
1	2
3	3
4	4
5	5
8	6
9	7

In [83]:

```
#23
df = pd.DataFrame(np.random.random(size=(5, 3)))
df.sub(df.mean(axis=1), axis=0)
```

Out[83]:

	0	1	2
0	0.013360	-0.283732	0.270372
1	-0.017856	0.134127	-0.116271
2	-0.061520	-0.377675	0.439195
3	-0.011979	-0.014577	0.026556
4	-0.283785	-0.194577	0.478362

In [85]:

```
#24
df = pd.DataFrame(np.random.random(size=(5, 10)), columns=list('abcdefghij'))
df
```

Out[85]:

	a	b	c	d	e	f	g	h	i	j
0	0.120036	0.986562	0.736506	0.315865	0.851590	0.457405	0.863789	0.916562	0.447701	0.691387
1	0.263504	0.903446	0.492755	0.646376	0.723349	0.830531	0.042265	0.737415	0.188006	0.999520
2	0.989898	0.100583	0.370534	0.483046	0.443577	0.501570	0.749686	0.329998	0.545060	0.961166
3	0.419724	0.342997	0.918084	0.468086	0.170632	0.217528	0.184905	0.421418	0.080474	0.995704
4	0.155133	0.877922	0.977828	0.371343	0.662908	0.757741	0.648685	0.639386	0.867361	0.973596

In [86]:

```
df.sum().idxmin()
```

Out[86]:

```
'a'
```

In [87]:

```
#25
len(df) - df.duplicated(keep=False).sum()
```

Out[87]:

```
5
```

In [88]:

```
#25
len(df.drop_duplicates(keep=False))
```

Out[88]:

```
5
```

In [89]:

```
#26
(df.isnull().cumsum(axis=1) == 3).idxmax(axis=1)
```

Out[89]:

```
0    a
1    a
2    a
3    a
4    a
dtype: object
```

In [100…

```
#27
df = pd.DataFrame({'grps': list('aaabbcaabcccbbc'),
                   'vals': [12,345,3,1,45,14,4,52,54,23,235,21,57,3,87]})
df.groupby('grps')['vals'].nlargest(3).sum(level=0)
```

```
C:\Users\swara\AppData\Local\Temp/ipykernel_24000/569070936.py:4: FutureWarning: Usin
g the level keyword in DataFrame and Series aggregations is deprecated and will be re
moved in a future version. Use groupby instead. df.sum(level=1) should use df.groupby
(level=1).sum().
  df.groupby('grps')['vals'].nlargest(3).sum(level=0)
```

Out[100…

```
grps
a    409
b    156
c    345
Name: vals, dtype: int64
```

In [103…

```
#28
df = pd.DataFrame(np.random.RandomState(8765).randint(1, 101, size=(100, 2)), columns
df
```

Out[103…

	A	B
0	46	29
1	75	22
2	49	63
3	33	43
4	71	75
...	...	...
95	60	87
96	57	40
97	86	19
98	50	56
99	97	94

100 rows × 2 columns

In [104…

```
df.groupby(pd.cut(df['A'], np.arange(0, 101, 10)))['B'].sum()
```

Out[104…

```
A
(0, 10]      635
(10, 20]     360
(20, 30]     315
(30, 40]     306
(40, 50]     750
(50, 60]     284
(60, 70]     424
(70, 80]     526
(80, 90]     835
(90, 100]    852
Name: B, dtype: int32
```

In [108…

```
#29
df = pd.DataFrame({'X': [7, 2, 0, 3, 4, 2, 5, 0, 3, 4]})
x = (df['X'] != 0).cumsum()
y = x != x.shift()
df['Y'] = y.groupby((y != y.shift()).cumsum()).cumsum()
df
```

Out[108…

	X	Y
0	7	1
1	2	2

2	0	0
3	3	1
4	4	2
5	2	3
6	5	4
7	0	0
8	3	1
9	4	2

In [111…

```
#30
df.unstack().sort_values()[-3:].index.tolist()
```

Out[111…

```
[('X', 4), ('X', 6), ('X', 0)]
```

In [125…

```
#31
df = pd.DataFrame({"vals": np.random.RandomState(31).randint(-30, 30, size=15),
                   "grps": np.random.RandomState(31).choice(["A", "B"], 15)})
df1 = df[df['vals'] == np.absolute(df['vals'])]
grp_mean = df1.groupby('grps')['vals'].mean()
#df[df['vals'] < 0]
df['patched_values']= df['vals'].mask(df['vals'] < 0, np.nan)
df['patched_values'] = df.apply(lambda row: grp_mean[row['grps']] if pd.isnull(row['p
                                 else row['patched_values'], axis=1)
df
```

Out[125…

	vals	grps	patched_values
0	-12	A	13.6
1	-7	B	28.0
2	-14	A	13.6
3	4	A	4.0
4	-7	A	13.6
5	28	B	28.0
6	-2	A	13.6
7	-1	A	13.6
8	8	A	8.0
9	-2	B	28.0
10	28	A	28.0
11	12	A	12.0
12	16	A	16.0
13	-24	A	13.6
14	-12	A	13.6

In [126…

```
#32
df = pd.DataFrame({'group': list('aabbabbbabab'),
                       'value': [1, 2, 3, np.nan, 2, 3, np.nan, 1, 7, 3, np.nan, 8]})
g1 = df.groupby(['group'])['value']               # group values
g2 = df.fillna(0).groupby(['group'])['value']     # fillna, then group values
```

```
s = g2.rolling(3, min_periods=1).sum() / g1.rolling(3, min_periods=1).count() # compu

s.reset_index(level=0, drop=True).sort_index()  # drop/sort index
```

Out[126…

```
0     1.000000
1     1.500000
2     3.000000
3     3.000000
4     1.666667
5     3.000000
6     3.000000
7     2.000000
8     3.666667
9     2.000000
10    4.500000
11    4.000000
Name: value, dtype: float64
```

In [124…

```
#33
dti = pd.date_range(start='2015-01-01', end='2015-12-31', freq='B')
s = pd.Series(np.random.rand(len(dti)), index=dti)
s
```

Out[124…

```
2015-01-01    0.978515
2015-01-02    0.831642
2015-01-05    0.608058
2015-01-06    0.704902
2015-01-07    0.037822
                ...
2015-12-25    0.006277
2015-12-28    0.317040
2015-12-29    0.960637
2015-12-30    0.565261
2015-12-31    0.778730
Freq: B, Length: 261, dtype: float64
```

In [132…

```
#34
s = pd.Series(np.random.rand(len(dti)), index=dti)
s[s.index.weekday == 2].sum()
```

Out[132…

```
24.38612066360484
```

In [133…

```
#35
s = pd.Series(np.random.rand(len(dti)), index=dti)
s.resample('M').mean()
```

Out[133…

```
2015-01-31    0.555097
2015-02-28    0.472481
2015-03-31    0.454140
2015-04-30    0.546419
2015-05-31    0.480409
2015-06-30    0.571384
2015-07-31    0.476889
2015-08-31    0.528675
2015-09-30    0.492234
2015-10-31    0.499429
2015-11-30    0.516382
2015-12-31    0.483872
Freq: M, dtype: float64
```

In [135…

```
#36
s = pd.Series(np.random.rand(len(dti)), index=dti)
s.groupby(pd.Grouper(freq='4M')).idxmax()
```

Out[135…

```
2015-01-31   2015-01-27
2015-05-31   2015-03-04
```

```
2015-09-30   2015-09-04
2016-01-31   2015-12-28
Freq: 4M, dtype: datetime64[ns]
```

In [136…

```
#37
pd.date_range('2015-01-01', '2016-12-31', freq='WOM-3THU')
```

Out[136…

```
DatetimeIndex(['2015-01-15', '2015-02-19', '2015-03-19', '2015-04-16',
               '2015-05-21', '2015-06-18', '2015-07-16', '2015-08-20',
               '2015-09-17', '2015-10-15', '2015-11-19', '2015-12-17',
               '2016-01-21', '2016-02-18', '2016-03-17', '2016-04-21',
               '2016-05-19', '2016-06-16', '2016-07-21', '2016-08-18',
               '2016-09-15', '2016-10-20', '2016-11-17', '2016-12-15'],
              dtype='datetime64[ns]', freq='WOM-3THU')
```

In [139…

```
#38
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_StockhOlm',
'Budapest_PaRis', 'Brussels_londOn'],
'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
'Airline': ['KLM(!)', '<Air France> (12)', '(British Airways. )',
'12. Air France', '"Swiss Air"']})
df
```

Out[139…

	From_To	FlightNumber	RecentDelays	Airline
0	LoNDon_paris	10045.0	[23, 47]	KLM(!)
1	MAdrid_miLAN	NaN	[]	<Air France> (12)
2	londON_StockhOlm	10065.0	[24, 43, 87]	(British Airways.)
3	Budapest_PaRis	NaN	[13]	12. Air France
4	Brussels_londOn	10085.0	[67, 32]	"Swiss Air"

In [140…

```
df['FlightNumber']
```

Out[140…

```
0    10045.0
1        NaN
2    10065.0
3        NaN
4    10085.0
Name: FlightNumber, dtype: float64
```

In [142…

```
newindex=np.arange(1,df.From_To.count()+1)
newindex
df.set_index(newindex, inplace=True)
df
```

Out[142…

	From_To	FlightNumber	RecentDelays	Airline
1	LoNDon_paris	10045.0	[23, 47]	KLM(!)
2	MAdrid_miLAN	NaN	[]	<Air France> (12)
3	londON_StockhOlm	10065.0	[24, 43, 87]	(British Airways.)
4	Budapest_PaRis	NaN	[13]	12. Air France
5	Brussels_londOn	10085.0	[67, 32]	"Swiss Air"

In [146…

```
df['FlightNumber'] = df['FlightNumber'].interpolate().astype(int)
df
```

Out[146…

	From_To	FlightNumber	RecentDelays	Airline
1	LoNDon_paris	10045	[23, 47]	KLM(!)
2	MAdrid_miLAN	10055	[]	<Air France> (12)
3	londON_StockhOlm	10065	[24, 43, 87]	(British Airways.)
4	Budapest_PaRis	10075	[13]	12. Air France
5	Brussels_londOn	10085	[67, 32]	"Swiss Air"

In [147…

```
df['FlightNumber'].astype(int)
```

Out[147…

```
1    10045
2    10055
3    10065
4    10075
5    10085
Name: FlightNumber, dtype: int32
```

In [149…

```
#39
temp = df.From_To.str.split('_', expand=True)
temp.columns = ['From', 'To']
temp
```

Out[149…

	From	To
1	LoNDon	paris
2	MAdrid	miLAN
3	londON	StockhOlm
4	Budapest	PaRis
5	Brussels	londOn

In [150…

```
#40
temp['From'] = temp['From'].str.capitalize()
temp['To'] = temp['To'].str.capitalize()
temp
```

Out[150…

	From	To
1	London	Paris
2	Madrid	Milan
3	London	Stockholm
4	Budapest	Paris
5	Brussels	London

In [166…

```
#41
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_StockhOlm',
'Budapest_PaRis', 'Brussels_londOn'],
'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
'Airline': ['KLM(!)', '<Air France> (12)', '(British Airways. )',
'12. Air France', '"Swiss Air"']})
df['FlightNumber'] = df['FlightNumber'].interpolate().astype(int)
df['From'] = ['London', 'Madrid', 'London', 'Budapest', 'Brussels']
df['To'] = ['Paris', 'Milan', 'Stockholm', 'Paris', 'London']
del df['From_To']
df
```

Out[166…

	FlightNumber	RecentDelays	Airline	From	To
0	10045	[23, 47]	KLM(!)	London	Paris
1	10055	[]	<Air France> (12)	Madrid	Milan
2	10065	[24, 43, 87]	(British Airways.)	London	Stockholm
3	10075	[13]	12. Air France	Budapest	Paris
4	10085	[67, 32]	"Swiss Air"	Brussels	London

In [168…

```
#42
df['Airline'] = df['Airline'].str.extract('([a-zA-Z\s]+)', expand=False).str.strip()
df
```

Out[168…

	FlightNumber	RecentDelays	Airline	From	To
0	10045	[23, 47]	KLM	London	Paris
1	10055	[]	Air France	Madrid	Milan
2	10065	[24, 43, 87]	British Airways	London	Stockholm
3	10075	[13]	Air France	Budapest	Paris
4	10085	[67, 32]	Swiss Air	Brussels	London

In [176…

```
#43
df = pd.DataFrame({'From_To': ['LoNDon_paris', 'MAdrid_miLAN', 'londON_StockhOlm',
'Budapest_PaRis', 'Brussels_londOn'],
'FlightNumber': [10045, np.nan, 10065, np.nan, 10085],
'RecentDelays': [[23, 47], [], [24, 43, 87], [13], [67, 32]],
'Airline': ['KLM(!)', '<Air France> (12)', '(British Airways. )',
'12. Air France', '"Swiss Air"']})
df['FlightNumber'] = df['FlightNumber'].interpolate().astype(int)
df['From'] = ['London', 'Madrid', 'London', 'Budapest', 'Brussels']
df['To'] = ['Paris', 'Milan', 'Stockholm', 'Paris', 'London']
del df['From_To']
delays = df['RecentDelays'].apply(pd.Series)

delays.columns = ['delay_{}'.format(n) for n in range(1, len(delays.columns)+1)]

df = df.drop('RecentDelays', axis=1).join(delays)
df
```

Out[176…

	FlightNumber	Airline	From	To	delay_1	delay_2	delay_3
0	10045	KLM(!)	London	Paris	23.0	47.0	NaN
1	10055	<Air France> (12)	Madrid	Milan	NaN	NaN	NaN
2	10065	(British Airways.)	London	Stockholm	24.0	43.0	87.0
3	10075	12. Air France	Budapest	Paris	13.0	NaN	NaN
4	10085	"Swiss Air"	Brussels	London	67.0	32.0	NaN

In [178…

```
#44
letters = ['A', 'B', 'C']
numbers = list(range(10))

mi = pd.MultiIndex.from_product([letters, numbers])
s = pd.Series(np.random.rand(30), index=mi)
s
```

```
Out[178…  A  0     0.579792
             1     0.323712
             2     0.067852
             3     0.846547
             4     0.275933
             5     0.484068
             6     0.947078
             7     0.001563
             8     0.225758
             9     0.131816
          B  0     0.479052
             1     0.258775
             2     0.603321
             3     0.724763
             4     0.087259
             5     0.904974
             6     0.354323
             7     0.939643
             8     0.075455
             9     0.934598
          C  0     0.194755
             1     0.502147
             2     0.176176
             3     0.061108
             4     0.430939
             5     0.419488
             6     0.789272
             7     0.789300
             8     0.283702
             9     0.904699
          dtype: float64
```

```
In [182…  #45
          s.index.is_lexsorted()
```

```
C:\Users\swara\AppData\Local\Temp/ipykernel_24000/1834326859.py:2: FutureWarning: Mul
tiIndex.is_lexsorted is deprecated as a public function, users should use MultiIndex.
is_monotonic_increasing instead.
  s.index.is_lexsorted()
Out[182…  True
```

```
In [183…  #46
          s.loc[:, [1, 3, 6]]
```

```
Out[183…  A  1     0.323712
             3     0.846547
             6     0.947078
          B  1     0.258775
             3     0.724763
             6     0.354323
          C  1     0.502147
             3     0.061108
             6     0.789272
          dtype: float64
```

```
In [184…  #47
          s.loc[pd.IndexSlice[:'B', 5:]]
```

```
Out[184…  A  5     0.484068
             6     0.947078
             7     0.001563
             8     0.225758
             9     0.131816
          B  5     0.904974
             6     0.354323
             7     0.939643
```

```
   8    0.075455
   9    0.934598
dtype: float64
```

In [187…

```
#48
s.sum(level=0)
```

```
C:\Users\swara\AppData\Local\Temp/ipykernel_24000/3962341031.py:2: FutureWarning: Usi
ng the level keyword in DataFrame and Series aggregations is deprecated and will be r
emoved in a future version. Use groupby instead. df.sum(level=1) should use df.groupb
y(level=1).sum().
  s.sum(level=0)
```

Out[187…

```
A    3.884119
B    5.362164
C    4.551587
dtype: float64
```

In [188…

```
#49
s.unstack().sum(axis=0)
```

Out[188…

```
0    1.253598
1    1.084634
2    0.847350
3    1.632418
4    0.794131
5    1.808530
6    2.090672
7    1.730507
8    0.584915
9    1.971114
dtype: float64
```

In [190…

```
#50
new_s = s.swaplevel(0, 1)

# check
new_s.index.is_lexsorted()

# sort
new_s = new_s.sort_index()
new_s
```

```
C:\Users\swara\AppData\Local\Temp/ipykernel_24000/2852934850.py:5: FutureWarning: Mul
tiIndex.is_lexsorted is deprecated as a public function, users should use MultiIndex.
is_monotonic_increasing instead.
  new_s.index.is_lexsorted()
```

Out[190…

```
0  A    0.579792
   B    0.479052
   C    0.194755
1  A    0.323712
   B    0.258775
   C    0.502147
2  A    0.067852
   B    0.603321
   C    0.176176
3  A    0.846547
   B    0.724763
   C    0.061108
4  A    0.275933
   B    0.087259
   C    0.430939
5  A    0.484068
   B    0.904974
   C    0.419488
6  A    0.947078
   B    0.354323
   C    0.789272
```

```
7  A    0.001563
   B    0.939643
   C    0.789300
8  A    0.225758
   B    0.075455
   C    0.283702
9  A    0.131816
   B    0.934598
   C    0.904699
dtype: float64
```

In [197…

```
#51
X = 5
Y = 4

p = pd.core.reshape.util.cartesian_product([np.arange(X), np.arange(Y)])
df = pd.DataFrame(np.asarray(p).T, columns=['x', 'y'])
df
```

Out[197…

	x	y
0	0	0
1	0	1
2	0	2
3	0	3
4	1	0
5	1	1
6	1	2
7	1	3
8	2	0
9	2	1
10	2	2
11	2	3
12	3	0
13	3	1
14	3	2
15	3	3
16	4	0
17	4	1
18	4	2
19	4	3

In [198…

```
#52
df['mine'] = np.random.binomial(1, 0.4, X*Y)
df
```

Out[198…

	x	y	mine
0	0	0	0
1	0	1	1
2	0	2	1

3	0	3	0
4	1	0	1
5	1	1	1
6	1	2	0
7	1	3	0
8	2	0	0
9	2	1	1
10	2	2	0
11	2	3	0
12	3	0	0
13	3	1	0
14	3	2	0
15	3	3	0
16	4	0	0
17	4	1	0
18	4	2	1
19	4	3	1

In [226…

```
#53
X = 5
Y = 4

p = pd.core.reshape.util.cartesian_product([np.arange(X), np.arange(Y)])
df = pd.DataFrame(np.asarray(p).T, columns=['x', 'y'])
df
df['adjacent'] = \
    df.merge(df + [ 1,  1], on=['x', 'y'], how='left')\
      .merge(df + [ 1, -1], on=['x', 'y'], how='left')\
      .merge(df + [-1,  1], on=['x', 'y'], how='left')\
      .merge(df + [-1, -1], on=['x', 'y'], how='left')\
      .merge(df + [ 1,  0], on=['x', 'y'], how='left')\
      .merge(df + [-1,  0], on=['x', 'y'], how='left')\
      .merge(df + [ 0,  1], on=['x', 'y'], how='left')\
      .merge(df + [ 0, -1], on=['x', 'y'], how='left')\
       .iloc[:, 3:]\
        .sum(axis=1)

df
```

Out[226…

	x	y	adjacent
0	0	0	0.0
1	0	1	0.0
2	0	2	0.0
3	0	3	0.0
4	1	0	0.0
5	1	1	0.0
6	1	2	0.0
7	1	3	0.0

8	2	0	0.0
9	2	1	0.0
10	2	2	0.0
11	2	3	0.0
12	3	0	0.0
13	3	1	0.0
14	3	2	0.0
15	3	3	0.0
16	4	0	0.0
17	4	1	0.0
18	4	2	0.0
19	4	3	0.0

In [205…

```
#54
df.loc[df['mine'] == 1, 'adjacent'] = np.nan
df
```

Out[205…

	x	y	mine	adjacent
0	0	0	0	3.0
1	0	1	1	NaN
2	0	2	1	NaN
3	0	3	0	1.0
4	1	0	1	NaN
5	1	1	1	NaN
6	1	2	0	4.0
7	1	3	0	1.0
8	2	0	0	3.0
9	2	1	1	NaN
10	2	2	0	2.0
11	2	3	0	0.0
12	3	0	0	1.0
13	3	1	0	2.0
14	3	2	0	3.0
15	3	3	0	2.0
16	4	0	0	0.0
17	4	1	0	1.0
18	4	2	1	NaN
19	4	3	1	NaN

In [206…

```
#55
df.drop('mine', axis=1)\
  .set_index(['y', 'x']).unstack()
```

Out[206…

	adjacent				
x	0	1	2	3	4
y					
0	3.0	NaN	3.0	1.0	0.0
1	NaN	NaN	NaN	2.0	1.0
2	NaN	4.0	2.0	3.0	NaN
3	1.0	1.0	0.0	2.0	NaN

In [207…

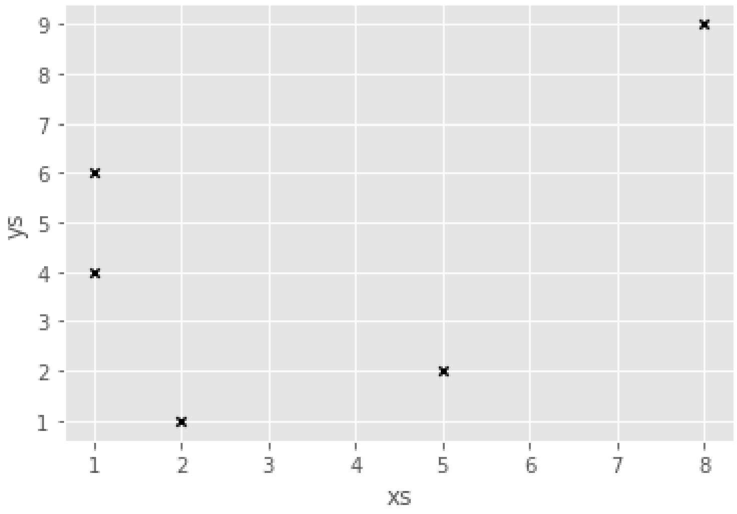
```
#56
import matplotlib.pyplot as plt
%matplotlib inline
plt.style.use('ggplot')

df = pd.DataFrame({"xs":[1,5,2,8,1], "ys":[4,2,1,9,6]})

df.plot.scatter("xs", "ys", color = "black", marker = "x")
```

Out[207…

```
<AxesSubplot:xlabel='xs', ylabel='ys'>
```



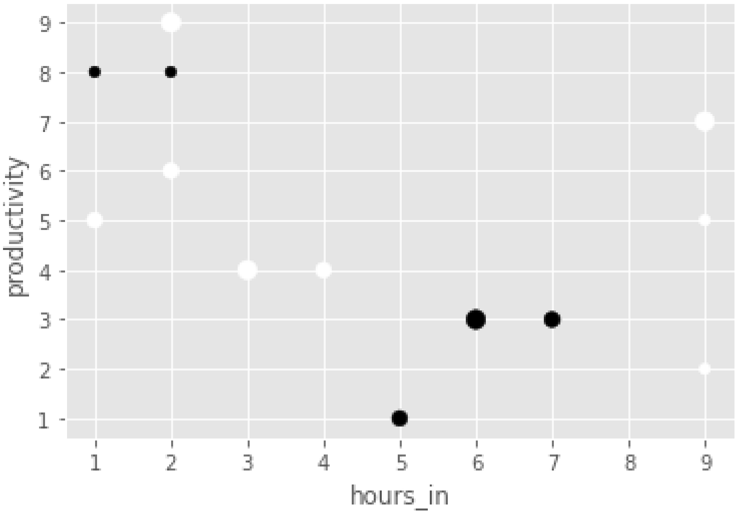
In [208…

```
#57
df = pd.DataFrame({"productivity":[5,2,3,1,4,5,6,7,8,3,4,8,9],
                   "hours_in"    :[1,9,6,5,3,9,2,9,1,7,4,2,2],
                   "happiness"   :[2,1,3,2,3,1,2,3,1,2,2,1,3],
                   "caffienated" :[0,0,1,1,0,0,0,0,1,1,0,1,0]})

df.plot.scatter("hours_in", "productivity", s = df.happiness * 30, c = df.caffienate
```

Out[208…

```
<AxesSubplot:xlabel='hours_in', ylabel='productivity'>
```



In [213…

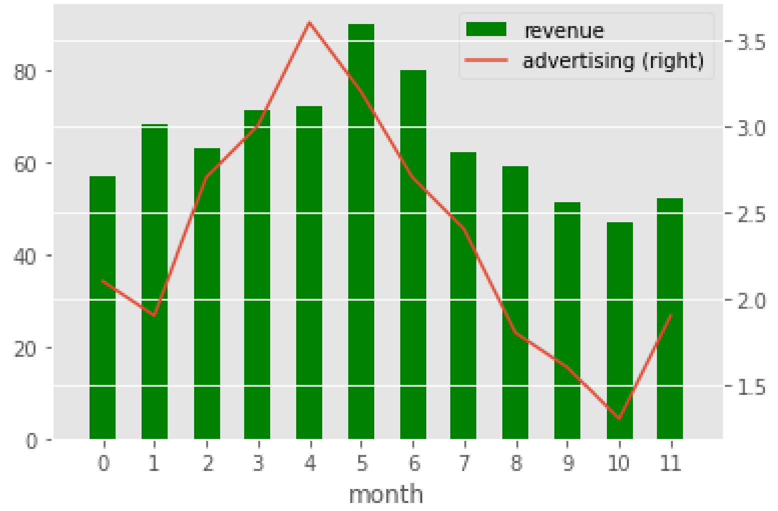
```
#58
df = pd.DataFrame({"revenue":[57,68,63,71,72,90,80,62,59,51,47,52],
                   "advertising":[2.1,1.9,2.7,3.0,3.6,3.2,2.7,2.4,1.8,1.6,1.3,1.9],
                   "month":range(12)
                  })

ax = df.plot.bar("month", "revenue", color = "green")
df.plot.line("month", "advertising", secondary_y = True, ax = ax)
ax.set_xlim((-1,12))
import numpy as np
def float_to_time(x):
    return str(int(x)) + ":" + str(int(x%1 * 60)).zfill(2) + ":" + str(int(x*60 % 1 *

def day_stock_data():
    #NYSE is open from 9:30 to 4:00
    time = 9.5
    price = 100
    results = [(float_to_time(time), price)]
    while time < 16:
        elapsed = np.random.exponential(.001)
        time += elapsed
        if time > 16:
            break
        price_diff = np.random.uniform(.999, 1.001)
        price *= price_diff
        results.append((float_to_time(time), price))


    df = pd.DataFrame(results, columns = ['time','price'])
    df.time = pd.to_datetime(df.time)
    return df
def plot_candlestick(agg):
    fig, ax = plt.subplots()
    for time in agg.index:
        ax.plot([time.hour] * 2, agg.loc[time, ["high","low"]].values, color = "black
        ax.plot([time.hour] * 2, agg.loc[time, ["open","close"]].values, color = agg.

    ax.set_xlim((8,16))
    ax.set_ylabel("Price")
    ax.set_xlabel("Hour")
    ax.set_title("OHLC of Stock Value During Trading Day")
    plt.show()
```



In [211…

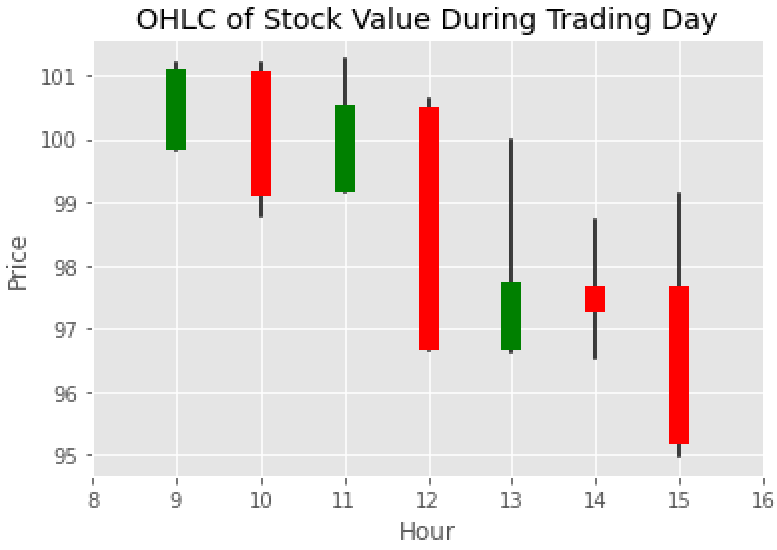
```
#59
df = day_stock_data()
df.head()
df.set_index("time", inplace = True)
agg = df.resample("H").ohlc()
agg.columns = agg.columns.droplevel()
agg["color"] = (agg.close > agg.open).map({True:"green",False:"red"})
agg.head()
```

Out[211…

	open	high	low	close	color
time					
2021-11-21 09:00:00	100.000000	101.194998	99.844406	100.950831	green
2021-11-21 10:00:00	100.912825	101.195331	98.776776	99.269324	red
2021-11-21 11:00:00	99.326435	101.249050	99.178432	100.375006	green
2021-11-21 12:00:00	100.327062	100.618391	96.645788	96.829507	red
2021-11-21 13:00:00	96.803384	99.983909	96.617814	97.575903	green

In [212…

```
#60
plot_candlestick(agg)
```



In [ ]: