How to iterate over rows in a DataFrame in Pandas

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

df = pd.DataFrame({'c1': [10, 11, 12], 'c2': [100, 110, 120]})
df

```

Out[1]:

	c1	c2
0	10	100
1	11	110
2	12	120

In [2]:
```
for index, row in df.iterrows():
    print(row['c1'], row['c2'])
```

```
10 100
11 110
12 120
```

In [3]:
```
for row in df.itertuples(index=True, name='Pandas'):
    print(row.c1, row.c2)
```

```
10 100
11 110
12 120
```

In [4]:
```
import numpy

df = pd.DataFrame({'a': numpy.random.randn(1000), 'b': numpy.random.randn(10

df
```

Out[4]:

	a	b	N	x
0	-0.470708	0.461152	524	x
1	0.014229	-0.018326	195	x
2	-0.832069	-0.167943	105	x
3	-1.003930	0.234902	390	x
4	-0.316543	1.228268	796	x
...	...	...	...	...
995	2.595775	0.209577	354	x
996	-2.119354	0.104997	859	x
997	-0.110201	-0.151901	707	x
998	-0.141059	-1.548599	832	x
999	0.594494	-1.560927	727	x

1000 rows × 4 columns

In [5]:
```
%timeit [row.a * 2 for idx, row in df.iterrows()]
# => 10 loops, best of 3: 50.3 ms per loop

%timeit [row[1] * 2 for row in df.itertuples()]
# => 1000 loops, best of 3: 541 µs per loop
```

```
87.2 ms ± 4.51 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
1.57 ms ± 201 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [6]:
```
df = pd.DataFrame({'c1': [10, 11, 12], 'c2': [100, 110, 120]})

#You can use the df.iloc function as follows:
for i in range(0, len(df)):
    print(df.iloc[i]['c1'], df.iloc[i]['c2'])
```

```
10 100
11 110
12 120
```

How to select rows from a DataFrame based on column values

In [7]:
```
import pandas as pd
import numpy as np
df = pd.DataFrame({'A': 'foo bar foo bar foo bar foo foo'.split(),
                   'B': 'one one two three two two one three'.split(),
                   'C': np.arange(8), 'D': np.arange(8) * 2})
print(df)
print(df.loc[df['A'] == 'foo'])

```

```
     A      B  C   D
0  foo    one  0   0
1  bar    one  1   2
2  foo    two  2   4
3  bar  three  3   6
4  foo    two  4   8
5  bar    two  5  10
6  foo    one  6  12
7  foo  three  7  14
     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

In [8]:
```
print(df.loc[df['B'].isin(['one','three'])])
```

```
     A      B  C   D
0  foo    one  0   0
1  bar    one  1   2
3  bar  three  3   6
6  foo    one  6  12
7  foo  three  7  14
```

In [9]:
```
# Note, however, that if you wish to do this many times, it is more efficien
df = df.set_index(['B'])
print(df.loc['one'])
```

```
       A  C   D
B
one  foo  0   0
one  bar  1   2
one  foo  6  12
```

In [10]:
```
# or, to include multiple values from the index use df.index.isin:

df.loc[df.index.isin(['one','two'])]
```

Out[10]:

B	A	C	D
one	foo	0	0
one	bar	1	2
two	foo	2	4
two	foo	4	8
two	bar	5	10
one	foo	6	12

In [11]:
```
import pandas as pd

# Create data set
d = {'foo':[100, 111, 222],
     'bar':[333, 444, 555]}
df = pd.DataFrame(d)

# Full dataframe:
df
```

Out[11]:

	foo	bar
0	100	333
1	111	444
2	222	555

In [12]:
```
df[df.foo == 222]
```

Out[12]:

	foo	bar
2	222	555

In [13]:
```
df[(df.foo == 222) | (df.bar == 444)]

```

Out[13]:

	foo	bar
1	111	444
2	222	555

In [14]:
```
df = pd.DataFrame(np.random.rand(10, 3), columns=list('abc'))
df
```

Out[14]:

	a	b	c
0	0.918417	0.957375	0.053473
1	0.278213	0.994802	0.316326
2	0.602847	0.025093	0.291699
3	0.240996	0.382499	0.101687
4	0.757964	0.126919	0.881387
5	0.626705	0.397941	0.238224
6	0.485094	0.210638	0.885264
7	0.908581	0.591107	0.676166
8	0.248287	0.459687	0.605169
9	0.609306	0.756531	0.838440

In [15]:
```
df[(df.a < df.b) & (df.b < df.c)]
```

Out[15]:

	a	b	c
8	0.248287	0.459687	0.605169
9	0.609306	0.756531	0.838440

In [16]:
```
df.query('(a < b) & (b < c)')
```

Out[16]:

	a	b	c
8	0.248287	0.459687	0.605169
9	0.609306	0.756531	0.838440

In [17]:
```
import pandas as pd
import numpy as np
df = pd.DataFrame({'A': 'foo bar foo bar foo bar foo foo'.split(),
                   'B': 'one one two three two two one three'.split(),
                   'C': np.arange(8), 'D': np.arange(8) * 2})
print(df)
print(df.loc[df['A'] == 'foo'])
df.iloc[np.where(df.A.values=='foo')]
```

```
     A      B  C   D
0  foo    one  0   0
1  bar    one  1   2
2  foo    two  2   4
3  bar  three  3   6
4  foo    two  4   8
5  bar    two  5  10
6  foo    one  6  12
7  foo  three  7  14
     A      B  C   D
0  foo    one  0   0
2  foo    two  2   4
4  foo    two  4   8
6  foo    one  6  12
7  foo  three  7  14
```

Out[17]:

	A	B	C	D
0	foo	one	0	0
2	foo	two	2	4
4	foo	two	4	8
6	foo	one	6	12
7	foo	three	7	14

In [18]:
```
%timeit df.iloc[np.where(df.A.values=='foo')]  # fastest
%timeit df.loc[df['A'] == 'foo']
%timeit df.loc[df['A'].isin(['foo'])]
%timeit df[df.A=='foo']
%timeit df.query('(A=="foo")')  # slowest
```

```
378 µs ± 26.9 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
541 µs ± 35.1 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
462 µs ± 4.03 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
615 µs ± 141 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
1.48 ms ± 49.3 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Renaming columns in pandas

In [19]:
```
df = pd.DataFrame({'$a':[1,2], '$b': [10,20]})
df
```

Out[19]:

	$a	$b
0	1	10
1	2	20

In [20]:
```
df.columns = ['a', 'b']
df
```

Out[20]:

	a	b
0	1	10
1	2	20

In [21]:
```
df = df.rename(columns={'oldName1': 'newName1', 'oldName2': 'newName2'})
# Or rename the existing DataFrame (rather than creating a copy)
df.rename(columns={'oldName1': 'newName1', 'oldName2': 'newName2'}, inplace=
```

In [22]:
```
df = pd.DataFrame('x', index=range(3), columns=list('abcde'))
df
```

Out[22]:

	a	b	c	d	e
0	x	x	x	x	x
1	x	x	x	x	x
2	x	x	x	x	x

In [23]:
```
df2 = df.rename({'a': 'X', 'b': 'Y'}, axis=1)  # new method
df2 = df.rename({'a': 'X', 'b': 'Y'}, axis='columns')
df2 = df.rename(columns={'a': 'X', 'b': 'Y'})  # old method

df2

```

Out[23]:

	X	Y	c	d	e
0	x	x	x	x	x
1	x	x	x	x	x
2	x	x	x	x	x

df.rename({'a': 'X', 'b': 'Y'}, axis=1, inplace=True) df

In [24]:
```
df2 = df.set_axis(['V', 'W', 'X', 'Y', 'Z'], axis=1, inplace=False)
df2
```

Out[24]:

	V	W	X	Y	Z
0	x	x	x	x	x
1	x	x	x	x	x
2	x	x	x	x	x

In [25]:
```
df.columns = ['V', 'W', 'X', 'Y', 'Z']
df
```

Out[25]:

	V	W	X	Y	Z
0	x	x	x	x	x
1	x	x	x	x	x
2	x	x	x	x	x

In [26]:
```
df.columns = df.columns.str.replace('$','')

```

Delete column from pandas DataFrame

In [27]:
```
df = pd.DataFrame({'A': 'foo bar foo bar foo bar foo foo'.split(),
                   'E': 'foo bar foo bar foo bar foo foo'.split(),
                   'F': 'foo bar foo bar foo bar foo foo'.split(),
                   'B': 'one one two three two two one three'.split(),
                   'C': np.arange(8), 'D': np.arange(8) * 2})
df
```

Out[27]:

	A	E	F	B	C	D
0	foo	foo	foo	one	0	0
1	bar	bar	bar	one	1	2
2	foo	foo	foo	two	2	4
3	bar	bar	bar	three	3	6
4	foo	foo	foo	two	4	8
5	bar	bar	bar	two	5	10
6	foo	foo	foo	one	6	12
7	foo	foo	foo	three	7	14

In [28]:
```
del df['A']
df
```

Out[28]:

	E	F	B	C	D
0	foo	foo	one	0	0
1	bar	bar	one	1	2
2	foo	foo	two	2	4
3	bar	bar	three	3	6
4	foo	foo	two	4	8
5	bar	bar	two	5	10
6	foo	foo	one	6	12
7	foo	foo	three	7	14

In [29]:
```
df = df.drop('B', 1)
df
```

Out[29]:

	E	F	C	D
0	foo	foo	0	0
1	bar	bar	1	2
2	foo	foo	2	4
3	bar	bar	3	6
4	foo	foo	4	8
5	bar	bar	5	10
6	foo	foo	6	12
7	foo	foo	7	14

In [30]:
```
df.drop('C', axis=1, inplace=True)
df
```

Out[30]:

	E	F	D
0	foo	foo	0
1	bar	bar	2
2	foo	foo	4
3	bar	bar	6
4	foo	foo	8
5	bar	bar	10
6	foo	foo	12
7	foo	foo	14

In [31]:
```
df.drop(['E', 'F'], axis=1, inplace=True)
df
```

Out[31]:

	D
0	0
1	2
2	4
3	6
4	8
5	10
6	12
7	14

In [32]:
```
df = pd.DataFrame({'A': 'foo bar foo bar foo bar foo foo'.split(),
                   'E': 'foo bar foo bar foo bar foo foo'.split(),
                   'F': 'foo bar foo bar foo bar foo foo'.split(),
                   'B': 'one one two three two two one three'.split(),
                   'C': np.arange(8), 'D': np.arange(8) * 2})
print(df)
#delete first, second, fourth
df.drop(df.columns[[0,1,3]], axis=1, inplace=True)
df
```

```
     A    E    F      B  C   D
0  foo  foo  foo    one  0   0
1  bar  bar  bar    one  1   2
2  foo  foo  foo    two  2   4
3  bar  bar  bar  three  3   6
4  foo  foo  foo    two  4   8
5  bar  bar  bar    two  5  10
6  foo  foo  foo    one  6  12
7  foo  foo  foo  three  7  14
```

Out[32]:

	F	C	D
0	foo	0	0
1	bar	1	2
2	foo	2	4
3	bar	3	6
4	foo	4	8
5	bar	5	10
6	foo	6	12
7	foo	7	14

Selecting multiple columns in a pandas dataframe

In [33]:
```
df = pd.DataFrame({'A': 'foo bar foo bar foo bar foo foo'.split(),
                   'E': 'foo bar foo bar foo bar foo foo'.split(),
                   'F': 'foo bar foo bar foo bar foo foo'.split(),
                   'B': 'one one two three two two one three'.split(),
                   'C': np.arange(8), 'D': np.arange(8) * 2})
df
```

Out[33]:

	A	E	F	B	C	D
0	foo	foo	foo	one	0	0
1	bar	bar	bar	one	1	2
2	foo	foo	foo	two	2	4
3	bar	bar	bar	three	3	6
4	foo	foo	foo	two	4	8
5	bar	bar	bar	two	5	10
6	foo	foo	foo	one	6	12
7	foo	foo	foo	three	7	14

In [34]:
```
df[['A','B']]
```

Out[34]:

	A	B
0	foo	one
1	bar	one
2	foo	two
3	bar	three
4	foo	two
5	bar	two
6	foo	one
7	foo	three

In [35]:
```
df.iloc[:, 0:2] # Remember that Python does not slice inclusive of the endin
```

Out[35]:

	A	E
0	foo	foo
1	bar	bar
2	foo	foo
3	bar	bar
4	foo	foo
5	bar	bar
6	foo	foo
7	foo	foo

In [36]:
```
#see the comma(,) , before comma is row and after comma is column
df.iloc[0:2, : ]
```

Out[36]:

	A	E	F	B	C	D
0	foo	foo	foo	one	0	0
1	bar	bar	bar	one	1	2

In [37]:
```
df1 = df.iloc[0, 0:2].copy() # To avoid the case where changing df1 also cha
df1
```

Out[37]:
```
A    foo
E    foo
Name: 0, dtype: object
```

In [38]:
```
df.loc[:, 'E':'B']
```

Out[38]:

	E	F	B
0	foo	foo	one
1	bar	bar	one
2	foo	foo	two
3	bar	bar	three
4	foo	foo	two
5	bar	bar	two
6	foo	foo	one
7	foo	foo	three

In [39]:
```
1 df.filter(['A', 'B'])
```

Out[39]:

	A	B
0	foo	one
1	bar	one
2	foo	two
3	bar	three
4	foo	two
5	bar	two
6	foo	one
7	foo	three

How do I get the row count of a pandas DataFrame?

In [40]:
```
1 df
```

Out[40]:

	A	E	F	B	C	D
0	foo	foo	foo	one	0	0
1	bar	bar	bar	one	1	2
2	foo	foo	foo	two	2	4
3	bar	bar	bar	three	3	6
4	foo	foo	foo	two	4	8
5	bar	bar	bar	two	5	10
6	foo	foo	foo	one	6	12
7	foo	foo	foo	three	7	14

In [41]:
```
1 df.shape
```

Out[41]: (8, 6)

In [42]:
```
1 len(df.index)
```

Out[42]: 8

In [43]:
```
1 df.count()
```

Out[43]:
```
A    8
E    8
F    8
B    8
C    8
D    8
dtype: int64
```

In [44]:
```
1 print(len(df.columns))
```

```
6
```

In [45]:
```
1 df.groupby('A').size()
```

Out[45]:
```
A
bar    3
foo    5
dtype: int64
```

In [46]:
```
1 df.groupby('A').count()
```

Out[46]:

A	E	F	B	C	D
bar	3	3	3	3	3
foo	5	5	5	5	5

Get list from pandas DataFrame column headers

In [47]:
```
1 list(df.columns.values)
```

Out[47]:
```
['A', 'E', 'F', 'B', 'C', 'D']
```

In [48]:
```
1 list(df)
```

Out[48]:
```
['A', 'E', 'F', 'B', 'C', 'D']
```

In [49]:
```
1 df.columns.values.tolist()
```

Out[49]:
```
['A', 'E', 'F', 'B', 'C', 'D']
```

In [50]:
```
1 df.columns.tolist()
```

Out[50]:
```
['A', 'E', 'F', 'B', 'C', 'D']
```

In [51]:
```
%timeit df.columns.tolist()
%timeit df.columns.values.tolist()
```

```
6.42 µs ± 199 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
644 ns ± 16.5 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [52]:
```
%timeit [column for column in df]
%timeit df.columns.values.tolist()
%timeit list(df)
%timeit list(df.columns.values)
```

```
8.84 µs ± 803 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
759 ns ± 117 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
11.4 µs ± 2.01 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
1.85 µs ± 58.9 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

In [53]:
```
[c for c in df]
```

Out[53]:
```
['A', 'E', 'F', 'B', 'C', 'D']
```

In [54]:
```
sorted(df)
```

Out[54]:
```
['A', 'B', 'C', 'D', 'E', 'F']
```

In [55]:
```
[*df]
```

Out[55]:
```
['A', 'E', 'F', 'B', 'C', 'D']
```

In [56]:
```
{*df}
```

Out[56]:
```
{'A', 'B', 'C', 'D', 'E', 'F'}
```

In [57]:
```
*df,
```

Out[57]:
```
('A', 'E', 'F', 'B', 'C', 'D')
```

In [58]:
```
*cols, = df
cols
```

Out[58]:
```
['A', 'E', 'F', 'B', 'C', 'D']
```

In [59]:
```
df.keys()
```

Out[59]:
```
Index(['A', 'E', 'F', 'B', 'C', 'D'], dtype='object')
```

Adding new column to existing DataFrame in Python pandas

In [60]:
```
df = pd.DataFrame({'A': 'foo bar foo bar foo bar foo foo'.split(),
                   'E': 'foo bar foo bar foo bar foo foo'.split(),
                   'F': 'foo bar foo bar foo bar foo foo'.split(),
                   'B': 'one one two three two two one three'.split(),
                   'C': np.arange(8), 'D': np.arange(8) * 2})
df
```

Out[60]:

	A	E	F	B	C	D
0	foo	foo	foo	one	0	0
1	bar	bar	bar	one	1	2
2	foo	foo	foo	two	2	4
3	bar	bar	bar	three	3	6
4	foo	foo	foo	two	4	8
5	bar	bar	bar	two	5	10
6	foo	foo	foo	one	6	12
7	foo	foo	foo	three	7	14

In [61]:
```
sLength = len(df['A'])
df['X'] = pd.Series(np.random.randn(sLength), index=df.index)
df
```

Out[61]:

	A	E	F	B	C	D	X
0	foo	foo	foo	one	0	0	-1.602983
1	bar	bar	bar	one	1	2	-0.551128
2	foo	foo	foo	two	2	4	0.931172
3	bar	bar	bar	three	3	6	0.543930
4	foo	foo	foo	two	4	8	-1.830537
5	bar	bar	bar	two	5	10	-0.557791
6	foo	foo	foo	one	6	12	1.544705
7	foo	foo	foo	three	7	14	-0.408825

In [62]:
```
df['X1'] = "X1"
df
```

Out[62]:

	A	E	F	B	C	D	X	X1
0	foo	foo	foo	one	0	0	-1.602983	X1
1	bar	bar	bar	one	1	2	-0.551128	X1
2	foo	foo	foo	two	2	4	0.931172	X1
3	bar	bar	bar	three	3	6	0.543930	X1
4	foo	foo	foo	two	4	8	-1.830537	X1
5	bar	bar	bar	two	5	10	-0.557791	X1
6	foo	foo	foo	one	6	12	1.544705	X1
7	foo	foo	foo	three	7	14	-0.408825	X1

In [63]:
```
df.loc[ : , 'new_col'] = "list_of_values"
df
```

Out[63]:

	A	E	F	B	C	D	X	X1	new_col
0	foo	foo	foo	one	0	0	-1.602983	X1	list_of_values
1	bar	bar	bar	one	1	2	-0.551128	X1	list_of_values
2	foo	foo	foo	two	2	4	0.931172	X1	list_of_values
3	bar	bar	bar	three	3	6	0.543930	X1	list_of_values
4	foo	foo	foo	two	4	8	-1.830537	X1	list_of_values
5	bar	bar	bar	two	5	10	-0.557791	X1	list_of_values
6	foo	foo	foo	one	6	12	1.544705	X1	list_of_values
7	foo	foo	foo	three	7	14	-0.408825	X1	list_of_values

In [64]:
```
df = pd.DataFrame({'a': [1, 2], 'b': [3, 4]})
df


```

Out[64]:

	a	b
0	1	3
1	2	4

In [65]:
```
df.assign(mean_a=df.a.mean(), mean_b=df.b.mean())
```

Out[65]:

	a	b	mean_a	mean_b
0	1	3	1.5	3.5
1	2	4	1.5	3.5

In [66]:
```
df['i'] = None
df
```

Out[66]:

	a	b	i
0	1	3	None
1	2	4	None

How to change the order of DataFrame columns?

In [67]:
```
df
```

Out[67]:

	a	b	i
0	1	3	None
1	2	4	None

In [68]:
```
df = df[['a', 'i', 'b']]
```

In [69]:
```
df
```

Out[69]:

	a	i	b
0	1	None	3
1	2	None	4

Create pandas Dataframe by appending one row at a time

In [70]:
```
import pandas as pd
from numpy.random import randint
```

In [71]:
```
df = pd.DataFrame(columns=['lib', 'qty1', 'qty2'])
```

In [72]:
```
for i in range(5):
    df.loc[i] = ['name' + str(i)] + list(randint(10, size=2))

df
```

Out[72]:

	lib	qty1	qty2
0	name0	1	0
1	name1	3	1
2	name2	7	0
3	name3	1	3
4	name4	3	8

In [73]:

```
import pandas as pd
import numpy as np
import time

# del df1, df2, df3, df4
numOfRows = 1000
# append
startTime = time.perf_counter()
df1 = pd.DataFrame(np.random.randint(100, size=(5,5)), columns=['A', 'B', 'C
for i in range( 1,numOfRows-4):
    df1 = df1.append( dict( (a,np.random.randint(100)) for a in ['A','B','C'
print('Elapsed time: {:6.3f} seconds for {:d} rows'.format(time.perf_counter
print(df1.shape)

# .loc w/o prealloc
startTime = time.perf_counter()
df2 = pd.DataFrame(np.random.randint(100, size=(5,5)), columns=['A', 'B', 'C
for i in range( 1,numOfRows):
    df2.loc[i]  = np.random.randint(100, size=(1,5))[0]
print('Elapsed time: {:6.3f} seconds for {:d} rows'.format(time.perf_counter
print(df2.shape)

# .loc with prealloc
df3 = pd.DataFrame(index=np.arange(0, numOfRows), columns=['A', 'B', 'C', 'D
startTime = time.perf_counter()
for i in range( 1,numOfRows):
    df3.loc[i]  = np.random.randint(100, size=(1,5))[0]
print('Elapsed time: {:6.3f} seconds for {:d} rows'.format(time.perf_counter
print(df3.shape)

# dict
startTime = time.perf_counter()
row_list = []
for i in range (0,5):
    row_list.append(dict( (a,np.random.randint(100)) for a in ['A','B','C','
for i in range( 1,numOfRows-4):
    dict1 = dict( (a,np.random.randint(100)) for a in ['A','B','C','D','E'])
    row_list.append(dict1)

df4 = pd.DataFrame(row_list, columns=['A','B','C','D','E'])
print('Elapsed time: {:6.3f} seconds for {:d} rows'.format(time.perf_counter
print(df4.shape)
```

```
Elapsed time:  0.910 seconds for 1000 rows
(1000, 5)
Elapsed time:  0.970 seconds for 1000 rows
(1000, 5)
Elapsed time:  0.321 seconds for 1000 rows
(1000, 5)
Elapsed time:  0.011 seconds for 1000 rows
(1000, 5)
```

Change column type in pandas

In [74]:
```
s = pd.Series(["8", 6, "7.5", 3, "0.9"]) # mixed string and numeric values
s
```

Out[74]:
```
0      8
1      6
2    7.5
3      3
4    0.9
dtype: object
```

In [75]:
```
pd.to_numeric(s) # convert everything to float values
```

Out[75]:
```
0    8.0
1    6.0
2    7.5
3    3.0
4    0.9
dtype: float64
```

In [76]:
```
df = pd.DataFrame({'A': '1 2 3 4 5 6 7 8'.split(),
                   'E': 'foo bar foo bar foo bar foo foo'.split(),
                   'F': 'foo bar foo bar foo bar foo foo'.split(),
                   'B': 'one one two three two two one three'.split(),
                   'C': np.arange(8), 'D': np.arange(8) * 2})
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 6 columns):
A    8 non-null object
E    8 non-null object
F    8 non-null object
B    8 non-null object
C    8 non-null int32
D    8 non-null int32
dtypes: int32(2), object(4)
memory usage: 448.0+ bytes
```

In [77]:
```
df["A"] = pd.to_numeric(df["A"])
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 6 columns):
A    8 non-null int64
E    8 non-null object
F    8 non-null object
B    8 non-null object
C    8 non-null int32
D    8 non-null int32
dtypes: int32(2), int64(1), object(3)
memory usage: 448.0+ bytes
```

In [78]:
```
s = pd.Series(['1', '2', '4.7', 'pandas', '10'])
s
```

Out[78]:
```
0         1
1         2
2       4.7
3    pandas
4        10
dtype: object
```

In [79]:
```
 pd.to_numeric(s, errors='coerce')
```

Out[79]:
```
0     1.0
1     2.0
2     4.7
3     NaN
4    10.0
dtype: float64
```

In [80]:
```
pd.to_numeric(s, errors='ignore')
```

Out[80]:
```
0         1
1         2
2       4.7
3    pandas
4        10
dtype: object
```

In [81]:
```
df.apply(pd.to_numeric, errors='ignore')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 6 columns):
A    8 non-null int64
E    8 non-null object
F    8 non-null object
B    8 non-null object
C    8 non-null int32
D    8 non-null int32
dtypes: int32(2), int64(1), object(3)
memory usage: 448.0+ bytes
```

In [82]:
```
s = pd.Series([1, 2, -7])
print(s)
print(pd.to_numeric(s, downcast='integer'))
print(pd.to_numeric(s, downcast='float'))
```

```
0    1
1    2
2   -7
dtype: int64
0    1
1    2
2   -7
dtype: int8
0    1.0
1    2.0
2   -7.0
dtype: float32
```

In [83]:
```
print(df.info())
# convert all DataFrame columns to the int64 dtype
df = df.astype(str)
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 6 columns):
A    8 non-null int64
E    8 non-null object
F    8 non-null object
B    8 non-null object
C    8 non-null int32
D    8 non-null int32
dtypes: int32(2), int64(1), object(3)
memory usage: 448.0+ bytes
None
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8 entries, 0 to 7
Data columns (total 6 columns):
A    8 non-null object
E    8 non-null object
F    8 non-null object
B    8 non-null object
C    8 non-null object
D    8 non-null object
dtypes: object(6)
memory usage: 512.0+ bytes
None
```

In [84]:
```
df = pd.DataFrame({'a': [7, 1, 5], 'b': ['3','2','1']}, dtype='object')
print(df.dtypes)
df = df.infer_objects()
print(df.dtypes)
```

```
a    object
b    object
dtype: object
a     int64
b    object
dtype: object
```

In [85]:
```
a = [['a', '1.2', '4.2'], ['b', '70', '0.03'], ['x', '5', '0']]
df = pd.DataFrame(a, columns=['one', 'two', 'three'])
print(df.dtypes)
df[['two', 'three']] = df[['two', 'three']].astype(float)
print(df.dtypes)
```

```
one      object
two      object
three    object
dtype: object
one       object
two      float64
three    float64
dtype: object
```

How to drop rows of Pandas DataFrame whose value in a certain column is NaN

In [86]:
```
df = pd.DataFrame(np.random.randn(10,3))
df
```

Out[86]:

	0	1	2
0	1.538227	-0.299883	-0.015679
1	-0.055671	-0.159953	0.620061
2	0.111928	-0.059591	0.618038
3	0.588351	-0.389214	0.523170
4	-1.189362	-1.863337	-0.051637
5	-0.942189	-0.373130	-0.312410
6	0.134624	-0.786627	0.698119
7	0.277452	2.480927	-0.834357
8	-0.013029	-1.145761	-0.657469
9	-0.871680	-0.698506	0.448499

In [87]:
```
df.iloc[::2,0] = np.nan; df.iloc[::4,1] = np.nan; df.iloc[::3,2] = np.nan;
df
```

Out[87]:

	0	1	2
0	NaN	NaN	NaN
1	-0.055671	-0.159953	0.620061
2	NaN	-0.059591	0.618038
3	0.588351	-0.389214	NaN
4	NaN	NaN	-0.051637
5	-0.942189	-0.373130	-0.312410
6	NaN	-0.786627	NaN
7	0.277452	2.480927	-0.834357
8	NaN	NaN	-0.657469
9	-0.871680	-0.698506	NaN

In [88]:
```
df.dropna()     #drop all rows that have any NaN values
```

Out[88]:

	0	1	2
1	-0.055671	-0.159953	0.620061
5	-0.942189	-0.373130	-0.312410
7	0.277452	2.480927	-0.834357

In [89]:
```
df.dropna(how='all')     #drop only if ALL columns are NaN
```

Out[89]:

	0	1	2
1	-0.055671	-0.159953	0.620061
2	NaN	-0.059591	0.618038
3	0.588351	-0.389214	NaN
4	NaN	NaN	-0.051637
5	-0.942189	-0.373130	-0.312410
6	NaN	-0.786627	NaN
7	0.277452	2.480927	-0.834357
8	NaN	NaN	-0.657469
9	-0.871680	-0.698506	NaN

In [90]:
```
1 df.dropna(thresh=2)   #Drop row if it does not have at least two values that
```

Out[90]:

	0	1	2
1	-0.055671	-0.159953	0.620061
2	NaN	-0.059591	0.618038
3	0.588351	-0.389214	NaN
5	-0.942189	-0.373130	-0.312410
7	0.277452	2.480927	-0.834357
9	-0.871680	-0.698506	NaN

In [91]:
```
1 df.dropna(subset=[1])   #Drop only if NaN in specific column (as asked in th
```

Out[91]:

	0	1	2
1	-0.055671	-0.159953	0.620061
2	NaN	-0.059591	0.618038
3	0.588351	-0.389214	NaN
5	-0.942189	-0.373130	-0.312410
6	NaN	-0.786627	NaN
7	0.277452	2.480927	-0.834357
9	-0.871680	-0.698506	NaN

In [92]:
```
1 df[pd.notnull(df[2])]
```

Out[92]:

	0	1	2
1	-0.055671	-0.159953	0.620061
2	NaN	-0.059591	0.618038
4	NaN	NaN	-0.051637
5	-0.942189	-0.373130	-0.312410
7	0.277452	2.480927	-0.834357
8	NaN	NaN	-0.657469

In [93]:
```
df[df[0].notnull()]
```

Out[93]:

	0	1	2
1	-0.055671	-0.159953	0.620061
3	0.588351	-0.389214	NaN
5	-0.942189	-0.373130	-0.312410
7	0.277452	2.480927	-0.834357
9	-0.871680	-0.698506	NaN

In [94]:
```
df[~df[0].notnull()]
```

Out[94]:

	0	1	2
0	NaN	NaN	NaN
2	NaN	-0.059591	0.618038
4	NaN	NaN	-0.051637
6	NaN	-0.786627	NaN
8	NaN	NaN	-0.657469

Use a list of values to select rows from a pandas dataframe

In [95]:
```
df = pd.DataFrame({'A': [5,6,3,4], 'B': [1,2,3,5]})
df
```

Out[95]:

	A	B
0	5	1
1	6	2
2	3	3
3	4	5

In [96]:
```
df[df['A'].isin([3, 6])]
```

Out[96]:

	A	B
1	6	2
2	3	3

In [97]:
```
df[~df['A'].isin([3, 6])]
```

Out[97]:

	A	B
0	5	1
3	4	5

How to deal with SettingWithCopyWarning in Pandas

In [98]:
```
df
```

Out[98]:

	A	B
0	5	1
1	6	2
2	3	3
3	4	5

In [99]:
```
df2 = df[['A']]
df2['A'] /= 2
```

```
D:\anaconda\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
```

In [100]:
```
df2 = df.loc[:, ['A']]
df2['A'] /= 2     # Does not raise
df2
```

Out[100]:

	A
0	2.5
1	3.0
2	1.5
3	2.0

In [101]:
```
pd.options.mode.chained_assignment = None
df2['A'] /= 2
df2
```

Out[101]:

	A
0	1.25
1	1.50
2	0.75
3	1.00

In [102]:
```
df2 = df[['A']].copy(deep=True)
df2['A'] /= 2
df2
```

Out[102]:

	A
0	2.5
1	3.0
2	1.5
3	2.0

In [103]:
```
#dropping a column on the copy may affect the original
data1 = {'A': [111, 112, 113], 'B':[121, 122, 123]}
df1 = pd.DataFrame(data1)
df1
```

Out[103]:

	A	B
0	111	121
1	112	122
2	113	123

In [104]:
```
df2 = df1
```

In [105]:
```
df2.drop('A', axis=1, inplace=True)
df1
```

Out[105]:

	B
0	121
1	122
2	123

In [106]:
```
#dropping a column on the original affects the copy
data1 = {'A': [111, 112, 113], 'B':[121, 122, 123]}
df1 = pd.DataFrame(data1)
df1
```

Out[106]:

	A	B
0	111	121
1	112	122
2	113	123

In [107]:
```
df2 = df1
df2.drop('A', axis=1, inplace=True)
df1
```

Out[107]:

	B
0	121
1	122
2	123

In [108]:
```
data1 = {'A': [111, 112, 113], 'B':[121, 122, 123]}
df1 = pd.DataFrame(data1)
df1
```

Out[108]:

	A	B
0	111	121
1	112	122
2	113	123

In [109]:
```
import copy
df2 = copy.deepcopy(df1)
df2
```

Out[109]:

	A	B
0	111	121
1	112	122
2	113	123

In [110]:
```
# Dropping a column on df1 does not affect df2
df2.drop('A', axis=1, inplace=True)
df1
```

Out[110]:

	A	B
0	111	121
1	112	122
2	113	123

Writing a pandas DataFrame to CSV file

In [111]:
```
# REFER BELOW LINK FOR MORE INFO
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFram
# df.to_csv(file_name, sep='\t',encoding='utf-8', index=False,header = False
#           date_format='%d/%m/%Y', na_rep='N/A')
# df.to_csv (r'C:\Users\John\Desktop\export_dataframe.csv', index = None, he
# df.to_csv(r'./export/dftocsv.csv', sep='\t', encoding='utf-8', header='tru
# df.to_dense().to_csv("submission.csv", index = False, sep=',', encoding='u
```

Convert list of dictionaries to a pandas DataFrame

In [112]:
```
d = [{'points': 50, 'time': '5:00', 'year': 2010},
{'points': 25, 'time': '6:00', 'month': "february"},
{'points':90, 'time': '9:00', 'month': 'january'},
{'points_h1':20, 'month': 'june'}]
d
```

Out[112]:
```
[{'points': 50, 'time': '5:00', 'year': 2010},
 {'points': 25, 'time': '6:00', 'month': 'february'},
 {'points': 90, 'time': '9:00', 'month': 'january'},
 {'points_h1': 20, 'month': 'june'}]
```

In [113]:
```
# The following methods all produce the same output.
print(pd.DataFrame(d))
print(pd.DataFrame.from_dict(d))
pd.DataFrame.from_records(d)
```

```
   points  time    year     month  points_h1
0    50.0  5:00  2010.0       NaN        NaN
1    25.0  6:00     NaN  february        NaN
2    90.0  9:00     NaN   january        NaN
3     NaN   NaN     NaN      june       20.0
   points  time    year     month  points_h1
0    50.0  5:00  2010.0       NaN        NaN
1    25.0  6:00     NaN  february        NaN
2    90.0  9:00     NaN   january        NaN
3     NaN   NaN     NaN      june       20.0
```

Out[113]:

	points	time	year	month	points_h1
0	50.0	5:00	2010.0	NaN	NaN
1	25.0	6:00	NaN	february	NaN
2	90.0	9:00	NaN	january	NaN
3	NaN	NaN	NaN	june	20.0

In [114]:
```
data_c = [
 {'A': 5, 'B': 0, 'C': 3, 'D': 3},
 {'A': 7, 'B': 9, 'C': 3, 'D': 5},
 {'A': 2, 'B': 4, 'C': 7, 'D': 6}]
pd.DataFrame.from_dict(data_c, orient='columns')
```

Out[114]:

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

In [115]:
```
data_i ={
 0: {'A': 5, 'B': 0, 'C': 3, 'D': 3},
 1: {'A': 7, 'B': 9, 'C': 3, 'D': 5},
 2: {'A': 2, 'B': 4, 'C': 7, 'D': 6}}
pd.DataFrame.from_dict(data_i, orient='index')
```

Out[115]:

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

Pretty-print an entire Pandas Series /

DataFrame

In [116]:
```
print(df)
```

```
   A  B
0  5  1
1  6  2
2  3  3
3  4  5
```

In [117]:
```
with pd.option_context('display.max_rows', None, 'display.max_columns', None
    print(df)
# with option_context('display.max_rows', 10, 'display.max_columns', 5):
#     print(df)

# pd.set_option('display.height',1000)
# pd.set_option('display.max_rows',500)
# pd.set_option('display.max_columns',500)
# pd.set_option('display.width',1000)
```

```
   A  B
0  5  1
1  6  2
2  3  3
3  4  5
```

In [118]:
```
print(df.to_string())
```

```
   A  B
0  5  1
1  6  2
2  3  3
3  4  5
```

In [119]:
```
pd.describe_option('display')
```

```
the screen width. The IPython notebook, IPython qtconsole, or IDLE
do not run in a terminal and hence it is not possible to do
correct auto-detection.
[default: 20] [currently: 20]display.max_colwidth : int
The maximum width in characters of a column in the repr of
a pandas data structure. When the column overflows, a "..."
placeholder is embedded in the output.
[default: 50] [currently: 50]display.max_info_columns : int
max_info_columns is used in DataFrame.info method to decide if
per column information will be printed.
[default: 100] [currently: 100]display.max_info_rows : int or None
df.info() will usually show null-counts for each column.
For large frames this can be quite slow. max_info_rows and max_info_cols
limit this null check only to frames with smaller dimensions than
specified.
[default: 1690785] [currently: 1690785]display.max_rows : int
If max_rows is exceeded, switch to truncate view. Depending on
`large_repr`, objects are either centrally truncated or printed as
a summary view. 'None' value means unlimited.
```

How do I expand the output display to see more columns of a pandas DataFrame?

In [120]:
```
# refer link below
# https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.set_opti
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

How are iloc and loc different?

In [121]:
```
s = pd.Series(list("abcdef"), index=[49, 48, 47, 0, 1, 2])
s
```

Out[121]:
```
49    a
48    b
47    c
0     d
1     e
2     f
dtype: object
```

In [122]:
```
# for loc and iloc , hint is observe the comma(,). before comma is row and a
```

In [123]:
```
s.loc[0]    # value at index label 0
```

Out[123]:
```
'd'
```

In [124]:
```
s.iloc[0]   # value at index location 0
```

Out[124]:
```
'a'
```

In [125]:
```
s.loc[0:1]  # rows at index labels between 0 and 1 (inclusive)
```

Out[125]:
```
0    d
1    e
dtype: object
```

In [126]:
```
s.iloc[0:1] # rows at index location between 0 and 1 (exclusive)
```

Out[126]:
```
49    a
dtype: object
```

In [127]:
```
df = pd.DataFrame(np.arange(25).reshape(5, 5),
                  index=list('abcde'),
                  columns=['x','y','z', 8, 9])
df
```

Out[127]:

	x	y	z	8	9
a	0	1	2	3	4
b	5	6	7	8	9
c	10	11	12	13	14
d	15	16	17	18	19
e	20	21	22	23	24

In [128]:
```
df.iloc[:df.index.get_loc('c') + 1, :4]
```

Out[128]:

	x	y	z	8
a	0	1	2	3
b	5	6	7	8
c	10	11	12	13

In [129]:
```
df.index.get_loc('c') + 1
```

Out[129]:
```
3
```

Deleting DataFrame row in Pandas based on column value

In [130]:
```
df
```

Out[130]:

	x	y	z	8	9
a	0	1	2	3	4
b	5	6	7	8	9
c	10	11	12	13	14
d	15	16	17	18	19
e	20	21	22	23	24

In [131]:
```
df2 = df[df.x != 0]
df2
```

Out[131]:

	x	y	z	8	9
b	5	6	7	8	9
c	10	11	12	13	14
d	15	16	17	18	19
e	20	21	22	23	24

In [132]:
```
df2 = df[df.x != None] #Doesn't do anything:
df2
```

Out[132]:

	x	y	z	8	9
a	0	1	2	3	4
b	5	6	7	8	9
c	10	11	12	13	14
d	15	16	17	18	19
e	20	21	22	23	24

In [133]:
```
df2 = df[df.x.notnull()]
df2
```

Out[133]:

	x	y	z	8	9
a	0	1	2	3	4
b	5	6	7	8	9
c	10	11	12	13	14
d	15	16	17	18	19
e	20	21	22	23	24

In [134]:
```
df.drop(df.loc[df['x']==0].index, inplace=True)
```

In [135]:
```
df
```

Out[135]:

	x	y	z	8	9
b	5	6	7	8	9
c	10	11	12	13	14
d	15	16	17	18	19
e	20	21	22	23	24

In [136]:
```
df[(df.x != 0) & (df.x != 10)]
```

Out[136]:

	x	y	z	8	9
b	5	6	7	8	9
d	15	16	17	18	19
e	20	21	22	23	24

In [137]:
```
df = df.drop(df[df['x']==0].index)
df
```

Out[137]:

	x	y	z	8	9
b	5	6	7	8	9
c	10	11	12	13	14
d	15	16	17	18	19
e	20	21	22	23	24

Combine two columns of text in pandas dataframe

In [138]:
```
df = pd.DataFrame(np.arange(25).reshape(5, 5),
                  index=list('abcde'),
                  columns=['x','y','z', 8, 9])
df
```

Out[138]:

	x	y	z	8	9
a	0	1	2	3	4
b	5	6	7	8	9
c	10	11	12	13	14
d	15	16	17	18	19
e	20	21	22	23	24

In [139]:
```
df["w"] = df["x"] + df["y"]
df
```

Out[139]:

	x	y	z	8	9	w
a	0	1	2	3	4	1
b	5	6	7	8	9	11
c	10	11	12	13	14	21
d	15	16	17	18	19	31
e	20	21	22	23	24	41

In [140]:
```
d = [{'points': 50, 'time': '5:00', 'year': 2010},
{'points': 25, 'time': '6:00', 'month': "february"},
{'points':90, 'time': '9:00', 'month': 'january'},
{'points_h1':20, 'month': 'june'}]

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

Out[140]:

	points	time	year	month	points_h1
0	50.0	5:00	2010.0	NaN	NaN
1	25.0	6:00	NaN	february	NaN
2	90.0	9:00	NaN	january	NaN
3	NaN	NaN	NaN	june	20.0

In [141]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 5 columns):
points       3 non-null float64
time         3 non-null object
year         1 non-null float64
month        3 non-null object
points_h1    1 non-null float64
dtypes: float64(3), object(2)
memory usage: 288.0+ bytes
```

In [142]:
```
df["w"] = df["points"].astype(str) + df["time"].astype(str)
df
```

Out[142]:

	points	time	year	month	points_h1	w
0	50.0	5:00	2010.0	NaN	NaN	50.05:00
1	25.0	6:00	NaN	february	NaN	25.06:00
2	90.0	9:00	NaN	january	NaN	90.09:00
3	NaN	NaN	NaN	june	20.0	nannan

In [143]:
```
df["w"] = df["points"] + df["year"]
df
```

Out[143]:

	points	time	year	month	points_h1	w
0	50.0	5:00	2010.0	NaN	NaN	2060.0
1	25.0	6:00	NaN	february	NaN	NaN
2	90.0	9:00	NaN	january	NaN	NaN
3	NaN	NaN	NaN	june	20.0	NaN

In [144]:
```
%timeit df['points'].astype(str) + df['time'].astype(str)

%timeit df['points'].map(str) + df['time'].map(str)

# %timeit df.points.str.cat(df.time.str)

%timeit df.loc[:, ['points','time']].astype(str).sum(axis=1)

%timeit df[['points','time']].astype(str).sum(axis=1)

%timeit df[['points','time']].apply(lambda x : '{}{}'.format(x[0],x[1]), axi
```

```
383 µs ± 18.1 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
712 µs ± 74.2 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
1.46 ms ± 104 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
1.47 ms ± 81.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
1.86 ms ± 46.7 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

Creating an empty Pandas DataFrame, then filling it?

In [145]:
```
import datetime
import pandas as pd
import numpy as np

todays_date = datetime.datetime.now().date()
todays_date
```

Out[145]:
```
datetime.date(2021, 2, 9)
```

In [146]:
```
index = pd.date_range(todays_date-datetime.timedelta(10), periods=10, freq='
index
```

Out[146]:
```
DatetimeIndex(['2021-01-30', '2021-01-31', '2021-02-01', '2021-02-02', '2021-02
-03', '2021-02-04', '2021-02-05', '2021-02-06', '2021-02-07', '2021-02-08'], dt
ype='datetime64[ns]', freq='D')
```

In [147]:
```
columns = ['A','B', 'C']
```

In [148]:
```
df_ = pd.DataFrame(index=index, columns=columns)
df_
```

Out[148]:

	A	B	C
2021-01-30	NaN	NaN	NaN
2021-01-31	NaN	NaN	NaN
2021-02-01	NaN	NaN	NaN
2021-02-02	NaN	NaN	NaN
2021-02-03	NaN	NaN	NaN
2021-02-04	NaN	NaN	NaN
2021-02-05	NaN	NaN	NaN
2021-02-06	NaN	NaN	NaN
2021-02-07	NaN	NaN	NaN
2021-02-08	NaN	NaN	NaN

In [149]:
```
df_ = df_.fillna(0) # with 0s rather than NaNs
df_
```

Out[149]:

	A	B	C
2021-01-30	0	0	0
2021-01-31	0	0	0
2021-02-01	0	0	0
2021-02-02	0	0	0
2021-02-03	0	0	0
2021-02-04	0	0	0
2021-02-05	0	0	0
2021-02-06	0	0	0
2021-02-07	0	0	0
2021-02-08	0	0	0

In [150]:
```
data = np.array([np.arange(10)]*3).T
data
```

Out[150]:
```
array([[0, 0, 0],
       [1, 1, 1],
       [2, 2, 2],
       [3, 3, 3],
       [4, 4, 4],
       [5, 5, 5],
       [6, 6, 6],
       [7, 7, 7],
       [8, 8, 8],
       [9, 9, 9]])
```

In [151]:
```
df = pd.DataFrame(data, index=index, columns=columns)
df
```

Out[151]:

	A	B	C
2021-01-30	0	0	0
2021-01-31	1	1	1
2021-02-01	2	2	2
2021-02-02	3	3	3
2021-02-03	4	4	4
2021-02-04	5	5	5
2021-02-05	6	6	6
2021-02-06	7	7	7
2021-02-07	8	8	8
2021-02-08	9	9	9

In [152]:
```
# Initialize empty frame with column names
col_names =  ['A', 'B', 'C']
my_df  = pd.DataFrame(columns = col_names)
my_df
```

Out[152]:

A	B	C

In [153]:
```
# Add a new record to a frame
my_df.loc[len(my_df)] = [2, 4, 5]
my_df
```

Out[153]:

	A	B	C
0	2	4	5

In [154]:
```
# You also might want to pass a dictionary:
my_dic = {'A':2, 'B':4, 'C':5}
my_df.loc[len(my_df)] = my_dic
my_df
```

Out[154]:

	A	B	C
0	2	4	5
1	2	4	5

In [155]:
```
# Append another frame to your existing frame
col_names =  ['A', 'B', 'C']
my_df2  = pd.DataFrame(columns = col_names)
my_df = my_df.append(my_df2)
my_df
```

Out[155]:

	A	B	C
0	2	4	5
1	2	4	5

Set value for particular cell in pandas DataFrame using index

In [156]:
```
df = pd.DataFrame(index=['A','B','C'], columns=['x','y'])
df
```

Out[156]:

	x	y
A	NaN	NaN
B	NaN	NaN
C	NaN	NaN

In [157]:
```
df.at['C', 'x'] = 10
df
```

Out[157]:

	x	y
A	NaN	NaN
B	NaN	NaN
C	10	NaN

In [158]:
```
df.set_value('C', 'x', 10)
df
```

```
D:\anaconda\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: set_value
is deprecated and will be removed in a future release. Please use .at[] or .iat
[] accessors instead
  """Entry point for launching an IPython kernel.
```

Out[158]:

	x	y
A	NaN	NaN
B	NaN	NaN
C	10	NaN

In [159]:
```
df['x']['C'] = 10
df
```

Out[159]:

	x	y
A	NaN	NaN
B	NaN	NaN
C	10	NaN

In [160]:
```
%timeit df.set_value('C', 'x', 10)
%timeit df['x']['C'] = 10
%timeit df.at['C', 'x'] = 10
```

```
D:\anaconda\lib\site-packages\ipykernel_launcher.py:1: FutureWarning: set_value
is deprecated and will be removed in a future release. Please use .at[] or .iat
[] accessors instead
  """Entry point for launching an IPython kernel.

4.57 µs ± 317 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
74.8 µs ± 6.52 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
6.13 µs ± 227 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

How to count the NaN values in a column in pandas DataFrame

In [161]:
```
s = pd.Series([1,2,3, np.nan, np.nan])
s
```

Out[161]:
```
0    1.0
1    2.0
2    3.0
3    NaN
4    NaN
dtype: float64
```

In [162]:
```
s.isna().sum()   # or s.isnull().sum() for older pandas versions
```

Out[162]:
```
2
```

In [163]:
```
df = pd.DataFrame({'a':[1,2,np.nan], 'b':[np.nan,1,np.nan]})
df
```

Out[163]:

	a	b
0	1.0	NaN
1	2.0	1.0
2	NaN	NaN

In [164]:
```
df.isna().sum()
```

Out[164]:
```
a    1
b    2
dtype: int64
```

In [165]:
```
count_nan = len(df) - df.count()
count_nan
```

Out[165]:
```
a    1
b    2
dtype: int64
```

In [166]:
```
df.isnull().sum(axis = 0) # This will give number of NaN values in every col
```

Out[166]:
```
a    1
b    2
dtype: int64
```

In [167]:
```
df.isnull().sum(axis = 1) # If you need, NaN values in every row,
```

Out[167]:
```
0    1
1    0
2    2
dtype: int64
```

In [168]:
```
df = pd.DataFrame({'a':[1,2,np.nan], 'b':[np.nan,1,np.nan]})
df
```

Out[168]:

	a	b
0	1.0	NaN
1	2.0	1.0
2	NaN	NaN

In [169]:
```
for col in df:
    print(df[col].value_counts(dropna=False))
```

```
NaN    1
2.0    1
1.0    1
Name: a, dtype: int64
NaN    2
1.0    1
Name: b, dtype: int64
```

In [170]:
```
1 df
```

Out[170]:

	a	b
0	1.0	NaN
1	2.0	1.0
2	NaN	NaN

In [171]:
```
1 df.isnull().sum().sort_values(ascending = False)
```

Out[171]:
```
b    2
a    1
dtype: int64
```

In [172]:
```
1 df.isnull().sum().sort_values(ascending = False).head(15) # The below will p
```

Out[172]:
```
b    2
a    1
dtype: int64
```

In [173]:
```
1 df.isnull().any().any()
```

Out[173]:
```
True
```

In [174]:
```
1  df.isnull().values.sum()
```

Out[174]:
```
3
```

In [175]:
```
1 df.isnull().any()
```

Out[175]:
```
a    True
b    True
dtype: bool
```

In [176]:
```
1 df.a.isnull().sum()
```

Out[176]:
```
1
```

In [177]:
```
1 df.b.isnull().sum()
```

Out[177]:
```
2
```

Select by partial string from a pandas DataFrame

In [178]:
```
d = [{'points': 50, 'time': '5:00', 'year': 2010},
{'points': 25, 'time': '6:00', 'month': "february"},
{'points':90, 'time': '9:00', 'month': 'january'},
{'points_h1':20, 'month': 'june'}]

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

Out[178]:

	points	time	year	month	points_h1
0	50.0	5:00	2010.0	NaN	NaN
1	25.0	6:00	NaN	february	NaN
2	90.0	9:00	NaN	january	NaN
3	NaN	NaN	NaN	june	20.0

In [179]:
```
df[df['month'].str.contains("janua",na=False)]
```

Out[179]:

	points	time	year	month	points_h1
2	90.0	9:00	NaN	january	NaN

In [180]:
```
df[df['month'].str.contains("janua|ne",na=False)]
```

Out[180]:

	points	time	year	month	points_h1
2	90.0	9:00	NaN	january	NaN
3	NaN	NaN	NaN	june	20.0

In [181]:
```
df[df['month'].str.contains(".*uary",na=False)]
```

Out[181]:

	points	time	year	month	points_h1
1	25.0	6:00	NaN	february	NaN
2	90.0	9:00	NaN	january	NaN

In [182]:
```
%timeit df[df['month'].str.contains('uary',na=False)]
%timeit df[df['month'].str.contains('uary', na=False,regex=False)]
```

```
552 µs ± 41.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
514 µs ± 38 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [183]:
```
s = pd.Series(['foo', 'foobar', np.nan, 'bar', 'baz', 123])
s
```

Out[183]:
```
0       foo
1    foobar
2       NaN
3       bar
4       baz
5       123
dtype: object
```

In [184]:
```
s.str.contains('foo|bar')

```

Out[184]:
```
0     True
1     True
2      NaN
3     True
4    False
5      NaN
dtype: object
```

In [185]:
```
s.str.contains('foo|bar', na=False)
```

Out[185]:
```
0     True
1     True
2    False
3     True
4    False
5    False
dtype: bool
```

In [186]:
```
df
```

Out[186]:

	points	time	year	month	points_h1
0	50.0	5:00	2010.0	NaN	NaN
1	25.0	6:00	NaN	february	NaN
2	90.0	9:00	NaN	january	NaN
3	NaN	NaN	NaN	june	20.0

In [187]:
```
df.filter(like='mon')  # select columns which contain the word mon
```

Out[187]:

	month
0	NaN
1	february
2	january
3	june

How to convert index of a pandas dataframe into a column?

In [188]:
```
d = [{'points': 50, 'time': '5:00', 'year': 2010},
{'points': 25, 'time': '6:00', 'month': "february"},
{'points':90, 'time': '9:00', 'month': 'january'},
{'points_h1':20, 'month': 'june'}]

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

Out[188]:

	points	time	year	month	points_h1
0	50.0	5:00	2010.0	NaN	NaN
1	25.0	6:00	NaN	february	NaN
2	90.0	9:00	NaN	january	NaN
3	NaN	NaN	NaN	june	20.0

In [189]:
```
df = df.rename_axis('index1').reset_index()
df
```

Out[189]:

	index1	points	time	year	month	points_h1
0	0	50.0	5:00	2010.0	NaN	NaN
1	1	25.0	6:00	NaN	february	NaN
2	2	90.0	9:00	NaN	january	NaN
3	3	NaN	NaN	NaN	june	20.0

In [190]:
```
df['index1'] = df.index
df
```

Out[190]:

	index1	points	time	year	month	points_h1
0	0	50.0	5:00	2010.0	NaN	NaN
1	1	25.0	6:00	NaN	february	NaN
2	2	90.0	9:00	NaN	january	NaN
3	3	NaN	NaN	NaN	june	20.0

In [191]:
```
# If you want to use the reset_index method and also preserve your existing
df.reset_index().set_index('index', drop=False)
```

Out[191]:

	index	index1	points	time	year	month	points_h1
index							
0	0	0	50.0	5:00	2010.0	NaN	NaN
1	1	1	25.0	6:00	NaN	february	NaN
2	2	2	90.0	9:00	NaN	january	NaN
3	3	3	NaN	NaN	NaN	june	20.0

Converting a Pandas GroupBy output from Series to DataFrame

In [192]:
```
df1 = pd.DataFrame( {
    "Name" : ["Alice", "Bob", "Mallory", "Mallory", "Bob" , "Mallory"] ,
    "City" : ["Seattle", "Seattle", "Portland", "Seattle", "Seattle", "Portl
df1
```

Out[192]:

	Name	City
0	Alice	Seattle
1	Bob	Seattle
2	Mallory	Portland
3	Mallory	Seattle
4	Bob	Seattle
5	Mallory	Portland

In [193]:
```
g1 = df1.groupby( [ "Name", "City"] ).count()
g1
```

Out[193]:

		Name	City
Alice	Seattle		
Bob	Seattle		
Mallory	Portland		
	Seattle		

In [194]:
```
type(g1)
```

Out[194]:
```
pandas.core.frame.DataFrame
```

In [195]:
```
g1.index
```

Out[195]:
```
MultiIndex([(  'Alice',  'Seattle'),
            (    'Bob',  'Seattle'),
            ('Mallory', 'Portland'),
            ('Mallory',  'Seattle')],
           names=['Name', 'City'])
```

In [196]:
```
g1.add_suffix('_Count').reset_index()
```

Out[196]:

	Name	City
0	Alice	Seattle
1	Bob	Seattle
2	Mallory	Portland
3	Mallory	Seattle

In [197]:
```
pd.DataFrame({'count' : df1.groupby( [ "Name", "City"] ).size()})
```

Out[197]:

		count
Name	City	
Alice	Seattle	1
Bob	Seattle	2
Mallory	Portland	2
	Seattle	1

Convert pandas dataframe to NumPy array

Convert pandas dataframe to NumPy array

In [198]:
```
import numpy as np
import pandas as pd

index = [1, 2, 3, 4, 5, 6, 7]
a = [np.nan, np.nan, np.nan, 0.1, 0.1, 0.1, 0.1]
b = [0.2, np.nan, 0.2, 0.2, 0.2, np.nan, np.nan]
c = [np.nan, 0.5, 0.5, np.nan, 0.5, 0.5, np.nan]
df = pd.DataFrame({'A': a, 'B': b, 'C': c}, index=index)
df = df.rename_axis('ID')
df
```

Out[198]:

ID	A	B	C
1	NaN	0.2	NaN
2	NaN	NaN	0.5
3	NaN	0.2	0.5
4	0.1	0.2	NaN
5	0.1	0.2	0.5
6	0.1	NaN	0.5
7	0.1	NaN	NaN

In [199]:
```
df.values
```

Out[199]:
```
array([[nan, 0.2, nan],
       [nan, nan, 0.5],
       [nan, 0.2, 0.5],
       [0.1, 0.2, nan],
       [0.1, 0.2, 0.5],
       [0.1, nan, 0.5],
       [0.1, nan, nan]])
```

In [200]:
```
df.to_numpy() # Convert the entire DataFrame
```

Out[200]:
```
array([[nan, 0.2, nan],
       [nan, nan, 0.5],
       [nan, 0.2, 0.5],
       [0.1, 0.2, nan],
       [0.1, 0.2, 0.5],
       [0.1, nan, 0.5],
       [0.1, nan, nan]])
```

In [201]:
```
df[['A', 'C']].to_numpy() # Convert specific columns
```

Out[201]:
```
array([[nan, nan],
       [nan, 0.5],
       [nan, 0.5],
       [0.1, nan],
       [0.1, 0.5],
       [0.1, 0.5],
       [0.1, nan]])
```

In [202]:
```
df.to_records() # If you need the dtypes in the result...
```

Out[202]:
```
rec.array([(1, nan, 0.2, nan), (2, nan, nan, 0.5), (3, nan, 0.2, 0.5),
           (4, 0.1, 0.2, nan), (5, 0.1, 0.2, 0.5), (6, 0.1, nan, 0.5),
           (7, 0.1, nan, nan)],
          dtype=[('ID', '<i8'), ('A', '<f8'), ('B', '<f8'), ('C', '<f8')])
```

How to filter Pandas dataframe using 'in' and 'not in' like in SQL

In [203]:
```
df1 = pd.DataFrame( {
    "Name" : ["Alice", "Bob", "Mallory", "Mallory", "Bob" , "Mallory"] ,
    "City" : ["Seattle", "hyderabad", "Portland", "Seattle", "Seattle", "Por
df1
```

Out[203]:

	Name	City
0	Alice	Seattle
1	Bob	hyderabad
2	Mallory	Portland
3	Mallory	Seattle
4	Bob	Seattle
5	Mallory	Portland

In [204]:
```
df1[df1.City.isin(['hyderabad','Portland'])]
```

Out[204]:

	Name	City
1	Bob	hyderabad
2	Mallory	Portland
5	Mallory	Portland

In [205]:
```
df1[~df1.City.isin(['hyderabad','Portland'])]
```

Out[205]:

	Name	City
0	Alice	Seattle
3	Mallory	Seattle
4	Bob	Seattle

In [206]:
```
df1.query("City in ('hyderabad','Portland')")
```

Out[206]:

	Name	City
1	Bob	hyderabad
2	Mallory	Portland
5	Mallory	Portland

In [207]:
```
df1.query("City not in ('hyderabad','Portland')")
```

Out[207]:

	Name	City
0	Alice	Seattle
3	Mallory	Seattle
4	Bob	Seattle

In [208]:
```
df = pd.DataFrame({'countries': ['US', 'UK', 'Germany', np.nan, 'China']})
df
```

Out[208]:

	countries
0	US
1	UK
2	Germany
3	NaN
4	China

In [209]:
```
c1 = ['UK', 'China']             # list
c2 = {'Germany'}                 # set
c3 = pd.Series(['China', 'US'])  # Series
c4 = np.array(['US', 'UK'])      # array
```

In [210]:
```
df[df['countries'].isin(c1)]
```

Out[210]:

	countries
1	UK
4	China

In [211]:
```
df[df['countries'].isin(c2)]
```

Out[211]:

	countries
2	Germany

In [212]:
```
df[df['countries'].isin(c3)]
```

Out[212]:

	countries
0	US
4	China

In [213]:
```
df[df['countries'].isin(c4)]
```

Out[213]:

	countries
0	US
1	UK

In [214]:
```
df2 = pd.DataFrame({
    'A': ['x', 'y', 'z', 'q'], 'B': ['w', 'a', np.nan, 'x'], 'C': np.arange(
df2
```

Out[214]:

	A	B	C
0	x	w	0
1	y	a	1
2	z	NaN	2
3	q	x	3

Shuffle DataFrame rows

In [215]:
```
1 df
```

Out[215]:

	countries
0	US
1	UK
2	Germany
3	NaN
4	China

In [216]:
```
1 df.sample(frac=1)
```

Out[216]:

	countries
2	Germany
0	US
3	NaN
1	UK
4	China

In [217]:
```
1 df = df.sample(frac=1).reset_index(drop=True)
2 df
```

Out[217]:

	countries
0	UK
1	NaN
2	China
3	US
4	Germany

In [218]:
```
from sklearn.utils import shuffle
df = shuffle(df)
df
```

Out[218]:

	countries
0	UK
2	China
4	Germany
1	NaN
3	US

Get statistics for each group (such as count, mean, etc) using pandas GroupBy?

In [219]:
```
data_i ={
 0: {'A': 5, 'B': 0, 'C': 3, 'D': 3},
 1: {'A': 7, 'B': 9, 'C': 3, 'D': 3},
 2: {'A': 2, 'B': 4, 'C': 7, 'D': 6}}
df = pd.DataFrame.from_dict(data_i, orient='index')
df
```

Out[219]:

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

In [220]:
```
df[['A', 'B', 'C', 'D']].groupby(['C', 'D']).agg(['mean', 'count'])
```

Out[220]:

		A		B	
		mean	count	mean	count
C	D				
3	3	6	2	4.5	2
7	6	2	1	4.0	1

In [221]:
```
df.groupby(['C','D']).size()
```

Out[221]:
```
C  D
3  3    2
7  6    1
dtype: int64
```

In [222]:
```
# Usually you want this result as a DataFrame (instead of a Series) so you c
df.groupby(['C','D']).size().reset_index(name='counts')
```

Out[222]:

	C	D	counts
0	3	3	2
1	7	6	1

In [223]:
```
df.groupby(['C','D'])['A'].describe()
```

Out[223]:

		count	mean	std	min	25%	50%	75%	max
C	D								
3	3	2.0	6.0	1.414214	5.0	5.5	6.0	6.5	7.0
7	6	1.0	2.0	NaN	2.0	2.0	2.0	2.0	2.0

In [224]:
```
df[['A','B','C','D']].groupby(['C','D']).count().reset_index()
```

Out[224]:

	C	D	A	B
0	3	3	2	2
1	7	6	1	1

How to replace NaN values by Zeroes in a column of a Pandas Dataframe?

In [225]:
```
df
```

Out[225]:

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

In [226]:
```
import numpy as np
import pandas as pd

index = [1, 2, 3, 4, 5, 6, 7]
a = [np.nan, np.nan, np.nan, 0.1, 0.1, 0.1, 0.1]
b = [0.2, np.nan, 0.2, 0.2, 0.2, np.nan, np.nan]
c = [np.nan, 0.5, 0.5, np.nan, 0.5, 0.5, np.nan]
df = pd.DataFrame({'A': a, 'B': b, 'C': c}, index=index)
df = df.rename_axis('ID')
df
```

Out[226]:

ID	A	B	C
1	NaN	0.2	NaN
2	NaN	NaN	0.5
3	NaN	0.2	0.5
4	0.1	0.2	NaN
5	0.1	0.2	0.5
6	0.1	NaN	0.5
7	0.1	NaN	NaN

In [227]:
```
df.fillna(0, inplace=True)
df
```

Out[227]:

ID	A	B	C
1	0.0	0.2	0.0
2	0.0	0.0	0.5
3	0.0	0.2	0.5
4	0.1	0.2	0.0
5	0.1	0.2	0.5
6	0.1	0.0	0.5
7	0.1	0.0	0.0

Difference between map, applymap and apply methods in Pandas

In [228]:
```
frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'), index=['Uta
frame
```

Out[228]:

	b	d	e
Utah	0.290525	1.352077	-0.163895
Ohio	0.674440	0.957184	-0.287462
Texas	-0.324501	0.080887	0.361652
Oregon	-1.937334	-1.997190	1.625434

In [229]:
```
f = lambda x: x.max() - x.min()
```

In [230]:
```
frame.apply(f)
```

Out[230]:
```
b    2.611773
d    3.349267
e    1.912896
dtype: float64
```

In [231]:
```
format = lambda x: '%.2f' % x
```

In [232]:
```
frame.applymap(format)
```

Out[232]:

	b	d	e
Utah	0.29	1.35	-0.16
Ohio	0.67	0.96	-0.29
Texas	-0.32	0.08	0.36
Oregon	-1.94	-2.00	1.63

In [233]:
```
frame['e'].map(format)
```

Out[233]:
```
Utah      -0.16
Ohio      -0.29
Texas      0.36
Oregon     1.63
Name: e, dtype: object
```

In [234]:
```
'''
map is defined on Series ONLY
applymap is defined on DataFrames ONLY
apply is defined on BOTH
'''
```

Out[234]:
```
'\nmap is defined on Series ONLY\napplymap is defined on DataFrames ONLY\napply is defined on BOTH\n'
```

UnicodeDecodeError when reading CSV file in Pandas with Python

read_csv takes an encoding option to deal with files in different formats. I mostly use read_csv('file', encoding = "ISO-8859-1"), or alternatively encoding = "utf-8" for reading, and generally utf-8 for to_csv

pd.read_csv('immigration.csv', encoding = "ISO-8859-1", engine='python')

Pandas Merging 101

In [235]:
```
np.random.seed(0)
left = pd.DataFrame({'key': ['A', 'B', 'C', 'D'], 'value': np.random.randn(4
right = pd.DataFrame({'key': ['B', 'D', 'E', 'F'], 'value': np.random.randn(
left
```

Out[235]:

	key	value
0	A	1.764052
1	B	0.400157
2	C	0.978738
3	D	2.240893

In [236]:
```
right
```

Out[236]:

	key	value
0	B	1.867558
1	D	-0.977278
2	E	0.950088
3	F	-0.151357

In [237]:
```
1 left.merge(right, on='key')
```

Out[237]:

	key	value_x	value_y
0	B	0.400157	1.867558
1	D	2.240893	-0.977278

In [238]:
```
1 left.merge(right, on='key', how='left')
```

Out[238]:

	key	value_x	value_y
0	A	1.764052	NaN
1	B	0.400157	1.867558
2	C	0.978738	NaN
3	D	2.240893	-0.977278

In [239]:
```
1 left.merge(right, on='key', how='right')
```

Out[239]:

	key	value_x	value_y
0	B	0.400157	1.867558
1	D	2.240893	-0.977278
2	E	NaN	0.950088
3	F	NaN	-0.151357

In [240]:
```
1 left.merge(right, on='key', how='outer')
```

Out[240]:

	key	value_x	value_y
0	A	1.764052	NaN
1	B	0.400157	1.867558
2	C	0.978738	NaN
3	D	2.240893	-0.977278
4	E	NaN	0.950088
5	F	NaN	-0.151357

In [241]:
```
(left.merge(right, on='key', how='left', indicator=True)
     .query('_merge == "left_only"')
     .drop('_merge', 1))
```

Out[241]:

	key	value_x	value_y
0	A	1.764052	NaN
2	C	0.978738	NaN

In [242]:
```
left.merge(right, on='key', how='left', indicator=True)
```

Out[242]:

	key	value_x	value_y	_merge
0	A	1.764052	NaN	left_only
1	B	0.400157	1.867558	both
2	C	0.978738	NaN	left_only
3	D	2.240893	-0.977278	both

In [243]:
```
(left.merge(right, on='key', how='right', indicator=True)
     .query('_merge == "right_only"')
     .drop('_merge', 1))
```

Out[243]:

	key	value_x	value_y
2	E	NaN	0.950088
3	F	NaN	-0.151357

In [244]:
```
(left.merge(right, on='key', how='outer', indicator=True)
     .query('_merge != "both"')
     .drop('_merge', 1))
```

Out[244]:

	key	value_x	value_y
0	A	1.764052	NaN
2	C	0.978738	NaN
4	E	NaN	0.950088
5	F	NaN	-0.151357

In [245]:
```
#Different names for key columns
left2 = left.rename({'key':'keyLeft'}, axis=1)
right2 = right.rename({'key':'keyRight'}, axis=1)
left2
```

Out[245]:

	keyLeft	value
0	A	1.764052
1	B	0.400157
2	C	0.978738
3	D	2.240893

In [246]:
```
right2
```

Out[246]:

	keyRight	value
0	B	1.867558
1	D	-0.977278
2	E	0.950088
3	F	-0.151357

In [247]:
```
left2.merge(right2, left_on='keyLeft', right_on='keyRight', how='inner')
```

Out[247]:

	keyLeft	value_x	keyRight	value_y
0	B	0.400157	B	1.867558
1	D	2.240893	D	-0.977278

In [248]:
```
#Avoiding duplicate key column in output
left3 = left2.set_index('keyLeft')
left3.merge(right2, left_index=True, right_on='keyRight')
```

Out[248]:

	value_x	keyRight	value_y
0	0.400157	B	1.867558
1	2.240893	D	-0.977278

In [249]:
```
#Merging on multiple columns
# left.merge(right, on=['key1', 'key2'] ...)
# Or, in the event the names are different,
# left.merge(right, left_on=['lkey1', 'lkey2'], right_on=['rkey1', 'rkey2'])
```

Import multiple csv files into pandas and concatenate into one DataFrame

In [250]:
```
'''
import pandas as pd
import glob

path = r'C:\DRO\DCL_rawdata_files' # use your path
all_files = glob.glob(path + "/*.csv")

li = []

for filename in all_files:
    df = pd.read_csv(filename, index_col=None, header=0)
    li.append(df)

frame = pd.concat(li, axis=0, ignore_index=True)
'''
```

Out[250]:
```
'\nimport pandas as pd\nimport glob\n\npath = r\'C:\\DRO\\DCL_rawdata_files\' # use your path\nall_files = glob.glob(path + "/*.csv")\n\nli = []\n\nfor filename in all_files:\n    df = pd.read_csv(filename, index_col=None, header=0)\n    li.append(df)\n\nframe = pd.concat(li, axis=0, ignore_index=True)\n'
```

In [251]:
```
'''path = r'C:\DRO\DCL_rawdata_files'                     # use your path
all_files = glob.glob(os.path.join(path, "*.csv"))     # advisable to use os

df_from_each_file = (pd.read_csv(f) for f in all_files)
concatenated_df   = pd.concat(df_from_each_file, ignore_index=True)
# doesn't create a list, nor does it append to one'''
```

Out[251]:
```
'path = r\'C:\\DRO\\DCL_rawdata_files\'                     # use your path\nall_files = glob.glob(os.path.join(path, "*.csv"))     # advisable to use os.path.join as this makes concatenation OS independent\n\ndf_from_each_file = (pd.read_csv(f) for f in all_files)\nconcatenated_df   = pd.concat(df_from_each_file, ignore_index=True)\n# doesn\'t create a list, nor does it append to one'
```

In [252]:
```
'''import glob
import os
import pandas as pd
df = pd.concat(map(pd.read_csv, glob.glob(os.path.join('', "my_files*.csv"))
```

Out[252]:
```
'import glob\nimport os\nimport pandas as pd   \ndf = pd.concat(map(pd.read_csv, glob.glob(os.path.join(\'\', "my_files*.csv"))))'
```

How to avoid Python/Pandas creating an index in a saved csv?

In [253]:
```
# df.to_csv('your.csv', index=False)
```

Filter dataframe rows if value in column is in a set list of values [duplicate]

In [254]:
```
# b = df[(df['a'] > 1) & (df['a'] < 5)]
```

Truth value of a Series is ambiguous. Use a.empty, a.bool(), a.item(), a.any() or a.all()

In [255]:
```
x = pd.Series([])
x.empty
```

Out[255]: True

In [256]:
```
x = pd.Series([1])
x.empty
```

Out[256]: False

In [257]:
```
x = pd.Series([100])
x
```

Out[257]:
```
0    100
dtype: int64
```

In [258]:
```
(x > 50).bool()
```

Out[258]: True

In [259]:
```
(x < 50).bool()
```

Out[259]: False

In [260]:
```
x = pd.Series([100])
x.item()
```

```
D:\anaconda\lib\site-packages\ipykernel_launcher.py:2: FutureWarning: `item` ha
s been deprecated and will be removed in a future version
```

Out[260]: 100

In [261]:
```
x = pd.Series([0, 1, 2])
x.all()   # because one element is zero
```

Out[261]: False

In [262]:
```
x.any()   # because one (or more) elements are non-zero
```

Out[262]: True

How to apply a function to two columns of

Pandas dataframe

In [263]:
```
import pandas as pd

df = pd.DataFrame({'ID':['1', '2', '3'], 'col_1': [0, 2, 3], 'col_2':[1, 4,
mylist = ['a', 'b', 'c', 'd', 'e', 'f']

def get_sublist(sta,end):
    return mylist[sta:end+1]

df['col_3'] = df.apply(lambda x: get_sublist(x.col_1, x.col_2), axis=1)
df
```

Out[263]:

	ID	col_1	col_2	col_3
0	1	0	1	[a, b]
1	2	2	4	[c, d, e]
2	3	3	5	[d, e, f]

How to get a value from a cell of a dataframe?

In [264]:
```
df
```

Out[264]:

	ID	col_1	col_2	col_3
0	1	0	1	[a, b]
1	2	2	4	[c, d, e]
2	3	3	5	[d, e, f]

In [265]:
```
df.iloc[0]
```

Out[265]:
```
ID            1
col_1         0
col_2         1
col_3    [a, b]
Name: 0, dtype: object
```

In [266]:
```
df.iloc[0]['col_2']
```

Out[266]: 1

Selecting a row of pandas series/dataframe by integer index

In [267]:
```
df = pd.DataFrame(np.random.rand(5,2),index=range(0,10,2),columns=list('AB'))
df
```

Out[267]:

	A	B
0	0.963663	0.383442
2	0.791725	0.528895
4	0.568045	0.925597
6	0.071036	0.087129
8	0.020218	0.832620

In [268]:
```
df.iloc[[2]]
```

Out[268]:

	A	B
4	0.568045	0.925597

In [269]:
```
df.loc[[2]]
```

Out[269]:

	A	B
2	0.791725	0.528895

How to pivot a dataframe?

In [270]:

```
import numpy as np
import pandas as pd
from numpy.core.defchararray import add

np.random.seed([3,1415])
n = 20

cols = np.array(['key', 'row', 'item', 'col'])
arr1 = (np.random.randint(5, size=(n, 4)) // [2, 1, 2, 1]).astype(str)

df = pd.DataFrame(
    add(cols, arr1), columns=cols
).join(
    pd.DataFrame(np.random.rand(n, 2).round(2)).add_prefix('val')
)

df
```

Out[270]:

	key	row	item	col	val0	val1
0	key0	row3	item1	col3	0.81	0.04
1	key1	row2	item1	col2	0.44	0.07
2	key1	row0	item1	col0	0.77	0.01
3	key0	row4	item0	col2	0.15	0.59
4	key1	row0	item2	col1	0.81	0.64
5	key1	row2	item2	col4	0.13	0.88
6	key2	row4	item1	col3	0.88	0.39
7	key1	row4	item1	col1	0.10	0.07
8	key1	row0	item2	col4	0.65	0.02
9	key1	row2	item0	col2	0.35	0.61
10	key2	row0	item2	col1	0.40	0.85
11	key2	row4	item1	col2	0.64	0.25
12	key0	row2	item2	col3	0.50	0.44
13	key0	row4	item1	col4	0.24	0.46
14	key1	row3	item2	col3	0.28	0.11
15	key0	row3	item1	col1	0.31	0.23
16	key0	row0	item2	col3	0.86	0.01
17	key0	row4	item0	col3	0.64	0.21
18	key2	row2	item2	col0	0.13	0.45
19	key0	row2	item0	col4	0.37	0.70

In [271]:
```
df.duplicated(['row', 'col']).any()
```

Out[271]: True

In [272]:
```
df.pivot_table(
    values='val0', index='row', columns='col',
    fill_value=0, aggfunc='mean')
```

Out[272]:

col	col0	col1	col2	col3	col4
row					
row0	0.77	0.605	0.000	0.860	0.65
row2	0.13	0.000	0.395	0.500	0.25
row3	0.00	0.310	0.000	0.545	0.00
row4	0.00	0.100	0.395	0.760	0.24

In [273]:
```
df.groupby(['row', 'col'])['val0'].mean().unstack(fill_value=0)
```

Out[273]:

col	col0	col1	col2	col3	col4
row					
row0	0.77	0.605	0.000	0.860	0.65
row2	0.13	0.000	0.395	0.500	0.25
row3	0.00	0.310	0.000	0.545	0.00
row4	0.00	0.100	0.395	0.760	0.24

In [274]:
```
pd.crosstab(
    index=df['row'], columns=df['col'],
    values=df['val0'], aggfunc='mean').fillna(0)
```

Out[274]:

col	col0	col1	col2	col3	col4
row					
row0	0.77	0.605	0.000	0.860	0.65
row2	0.13	0.000	0.395	0.500	0.25
row3	0.00	0.310	0.000	0.545	0.00
row4	0.00	0.100	0.395	0.760	0.24

In [275]:
```
df.pivot_table(
    values='val0', index='row', columns='col',
    fill_value=0, aggfunc='sum')
```

Out[275]:

col	col0	col1	col2	col3	col4
row					
row0	0.77	1.21	0.00	0.86	0.65
row2	0.13	0.00	0.79	0.50	0.50
row3	0.00	0.31	0.00	1.09	0.00
row4	0.00	0.10	0.79	1.52	0.24

In [276]:
```
df.groupby(['row', 'col'])['val0'].sum().unstack(fill_value=0)
```

Out[276]:

col	col0	col1	col2	col3	col4
row					
row0	0.77	1.21	0.00	0.86	0.65
row2	0.13	0.00	0.79	0.50	0.50
row3	0.00	0.31	0.00	1.09	0.00
row4	0.00	0.10	0.79	1.52	0.24

In [277]:
```
pd.crosstab(
    index=df['row'], columns=df['col'],
    values=df['val0'], aggfunc='sum').fillna(0)
```

Out[277]:

col	col0	col1	col2	col3	col4
row					
row0	0.77	1.21	0.00	0.86	0.65
row2	0.13	0.00	0.79	0.50	0.50
row3	0.00	0.31	0.00	1.09	0.00
row4	0.00	0.10	0.79	1.52	0.24

In [278]:
```
df.pivot_table(
    values='val0', index='row', columns='col',
    fill_value=0, aggfunc=[np.size, np.mean])
```

Out[278]:

	size					mean				
col	col0	col1	col2	col3	col4	col0	col1	col2	col3	col4
row										
row0	1	2	0	1	1	0.77	0.605	0.000	0.860	0.65
row2	1	0	2	1	2	0.13	0.000	0.395	0.500	0.25
row3	0	1	0	2	0	0.00	0.310	0.000	0.545	0.00
row4	0	1	2	2	1	0.00	0.100	0.395	0.760	0.24

In [279]:
```
df.groupby(['row', 'col'])['val0'].agg(['size', 'mean']).unstack(fill_value=
```

Out[279]:

	size					mean				
col	col0	col1	col2	col3	col4	col0	col1	col2	col3	col4
row										
row0	1	2	0	1	1	0.77	0.605	0.000	0.860	0.65
row2	1	0	2	1	2	0.13	0.000	0.395	0.500	0.25
row3	0	1	0	2	0	0.00	0.310	0.000	0.545	0.00
row4	0	1	2	2	1	0.00	0.100	0.395	0.760	0.24

In [280]:
```
pd.crosstab(
    index=df['row'], columns=df['col'],
    values=df['val0'], aggfunc=[np.size, np.mean]).fillna(0, downcast='infer
```

Out[280]:

	size					mean				
col	col0	col1	col2	col3	col4	col0	col1	col2	col3	col4
row										
row0	1	2	0	1	1	0.77	0.605	0.000	0.860	0.65
row2	1	0	2	1	2	0.13	0.000	0.395	0.500	0.25
row3	0	1	0	2	0	0.00	0.310	0.000	0.545	0.00
row4	0	1	2	2	1	0.00	0.100	0.395	0.760	0.24

In [281]:
```
df.pivot_table(
    values=['val0', 'val1'], index='row', columns='col',
    fill_value=0, aggfunc='mean')
```

Out[281]:

	val0					val1				
col	col0	col1	col2	col3	col4	col0	col1	col2	col3	col4
row										
row0	0.77	0.605	0.000	0.860	0.65	0.01	0.745	0.00	0.010	0.02
row2	0.13	0.000	0.395	0.500	0.25	0.45	0.000	0.34	0.440	0.79
row3	0.00	0.310	0.000	0.545	0.00	0.00	0.230	0.00	0.075	0.00
row4	0.00	0.100	0.395	0.760	0.24	0.00	0.070	0.42	0.300	0.46

In [282]:
```
df.pivot_table(
    values='val0', index='row', columns=['item', 'col'],
    fill_value=0, aggfunc='mean')
```

Out[282]:

item	item0			item1					item2			
col	col2	col3	col4	col0	col1	col2	col3	col4	col0	col1	col3	col4
row												
row0	0.00	0.00	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.605	0.86	0.65
row2	0.35	0.00	0.37	0.00	0.00	0.44	0.00	0.00	0.13	0.000	0.50	0.13
row3	0.00	0.00	0.00	0.00	0.31	0.00	0.81	0.00	0.00	0.000	0.28	0.00
row4	0.15	0.64	0.00	0.00	0.10	0.64	0.88	0.24	0.00	0.000	0.00	0.00

In [283]:
```
df.groupby(
    ['row', 'item', 'col']
)['val0'].mean().unstack(['item', 'col']).fillna(0).sort_index(1)
```

Out[283]:

item	item0			item1					item2			
col	col2	col3	col4	col0	col1	col2	col3	col4	col0	col1	col3	col4
row												
row0	0.00	0.00	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.605	0.86	0.65
row2	0.35	0.00	0.37	0.00	0.00	0.44	0.00	0.00	0.13	0.000	0.50	0.13
row3	0.00	0.00	0.00	0.00	0.31	0.00	0.81	0.00	0.00	0.000	0.28	0.00
row4	0.15	0.64	0.00	0.00	0.10	0.64	0.88	0.24	0.00	0.000	0.00	0.00

In [284]:

```
df.pivot_table(
    values='val0', index=['key', 'row'], columns=['item', 'col'],
    fill_value=0, aggfunc='mean')
```

Out[284]:

	item	item0			item1					item2			
	col	col2	col3	col4	col0	col1	col2	col3	col4	col0	col1	col3	col4
key	row												
key0	row0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.00
	row2	0.00	0.00	0.37	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.00
	row3	0.00	0.00	0.00	0.00	0.31	0.00	0.81	0.00	0.00	0.00	0.00	0.00
	row4	0.15	0.64	0.00	0.00	0.00	0.00	0.00	0.24	0.00	0.00	0.00	0.00
key1	row0	0.00	0.00	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.81	0.00	0.65
	row2	0.35	0.00	0.00	0.00	0.00	0.44	0.00	0.00	0.00	0.00	0.00	0.13
	row3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.28	0.00
	row4	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00
key2	row0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	0.00	0.00
	row2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	0.00	0.00	0.00
	row4	0.00	0.00	0.00	0.00	0.00	0.64	0.88	0.00	0.00	0.00	0.00	0.00

In [285]:
```
df.groupby(
    ['key', 'row', 'item', 'col']
)['val0'].mean().unstack(['item', 'col']).fillna(0).sort_index(1)
```

Out[285]:

	item	item0			item1					item2			
	col	col2	col3	col4	col0	col1	col2	col3	col4	col0	col1	col3	col4
key	row												
key0	row0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.00
	row2	0.00	0.00	0.37	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.00
	row3	0.00	0.00	0.00	0.00	0.31	0.00	0.81	0.00	0.00	0.00	0.00	0.00
	row4	0.15	0.64	0.00	0.00	0.00	0.00	0.00	0.24	0.00	0.00	0.00	0.00
key1	row0	0.00	0.00	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.81	0.00	0.65
	row2	0.35	0.00	0.00	0.00	0.00	0.44	0.00	0.00	0.00	0.00	0.00	0.13
	row3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.28	0.00
	row4	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00
key2	row0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	0.00	0.00
	row2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	0.00	0.00	0.00
	row4	0.00	0.00	0.00	0.00	0.00	0.64	0.88	0.00	0.00	0.00	0.00	0.00

In [286]:
```
df.set_index(
    ['key', 'row', 'item', 'col']
)['val0'].unstack(['item', 'col']).fillna(0).sort_index(1)
```

Out[286]:

	item	item0			item1					item2			
	col	col2	col3	col4	col0	col1	col2	col3	col4	col0	col1	col3	col4
key	row												
key0	row0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.86	0.00
	row2	0.00	0.00	0.37	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.50	0.00
	row3	0.00	0.00	0.00	0.00	0.31	0.00	0.81	0.00	0.00	0.00	0.00	0.00
	row4	0.15	0.64	0.00	0.00	0.00	0.00	0.00	0.24	0.00	0.00	0.00	0.00
key1	row0	0.00	0.00	0.00	0.77	0.00	0.00	0.00	0.00	0.00	0.81	0.00	0.65
	row2	0.35	0.00	0.00	0.00	0.00	0.44	0.00	0.00	0.00	0.00	0.00	0.13
	row3	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.28	0.00
	row4	0.00	0.00	0.00	0.00	0.10	0.00	0.00	0.00	0.00	0.00	0.00	0.00
key2	row0	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.40	0.00	0.00
	row2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	0.00	0.00	0.00
	row4	0.00	0.00	0.00	0.00	0.00	0.64	0.88	0.00	0.00	0.00	0.00	0.00

In [287]:
```
1 df.pivot_table(index='row', columns='col', fill_value=0, aggfunc='size')
```

Out[287]:

col	col0	col1	col2	col3	col4
row					
row0	1	2	0	1	1
row2	1	0	2	1	2
row3	0	1	0	2	0
row4	0	1	2	2	1

In [288]:
```
1 df.groupby(['row', 'col'])['val0'].size().unstack(fill_value=0)
```

Out[288]:

col	col0	col1	col2	col3	col4
row					
row0	1	2	0	1	1
row2	1	0	2	1	2
row3	0	1	0	2	0
row4	0	1	2	2	1

In [289]:
```
1 pd.crosstab(df['row'], df['col'])
```

Out[289]:

col	col0	col1	col2	col3	col4
row					
row0	1	2	0	1	1
row2	1	0	2	1	2
row3	0	1	0	2	0
row4	0	1	2	2	1

In [290]:
```
# get integer factorization `i` and unique values `r`
# for column `'row'`
i, r = pd.factorize(df['row'].values)
# get integer factorization `j` and unique values `c`
# for column `'col'`
j, c = pd.factorize(df['col'].values)
# `n` will be the number of rows
# `m` will be the number of columns
n, m = r.size, c.size
# `i * m + j` is a clever way of counting the
# factorization bins assuming a flat array of length
# `n * m`.  Which is why we subsequently reshape as `(n, m)`
b = np.bincount(i * m + j, minlength=n * m).reshape(n, m)
# BTW, whenever I read this, I think 'Bean, Rice, and Cheese'
pd.DataFrame(b, r, c)
```

Out[290]:

	col3	col2	col0	col1	col4
row3	2	0	0	1	0
row2	1	2	1	0	2
row0	1	0	1	2	1
row4	2	2	0	1	1

In [291]:
```
pd.get_dummies(df['row']).T.dot(pd.get_dummies(df['col']))
```

Out[291]:

	col0	col1	col2	col3	col4
row0	1	2	0	1	1
row2	1	0	2	1	2
row3	0	1	0	2	0
row4	0	1	2	2	1

In [292]:
```
df.columns = df.columns.map('|'.join)
df
```

Out[292]:

	k|e|y	r|o|w	i|t|e|m	c|o|l	v|a|l|0	v|a|l|1
0	key0	row3	item1	col3	0.81	0.04
1	key1	row2	item1	col2	0.44	0.07
2	key1	row0	item1	col0	0.77	0.01
3	key0	row4	item0	col2	0.15	0.59
4	key1	row0	item2	col1	0.81	0.64
5	key1	row2	item2	col4	0.13	0.88
6	key2	row4	item1	col3	0.88	0.39
7	key1	row4	item1	col1	0.10	0.07
8	key1	row0	item2	col4	0.65	0.02
9	key1	row2	item0	col2	0.35	0.61
10	key2	row0	item2	col1	0.40	0.85
11	key2	row4	item1	col2	0.64	0.25
12	key0	row2	item2	col3	0.50	0.44
13	key0	row4	item1	col4	0.24	0.46
14	key1	row3	item2	col3	0.28	0.11
15	key0	row3	item1	col1	0.31	0.23
16	key0	row0	item2	col3	0.86	0.01
17	key0	row4	item0	col3	0.64	0.21
18	key2	row2	item2	col0	0.13	0.45
19	key0	row2	item0	col4	0.37	0.70

In [293]:
```
df.columns = df.columns.map('{0[0]}|{0[1]}'.format)

df
```

Out[293]:

	k||	r||	i||	c||	v||	v||
0	key0	row3	item1	col3	0.81	0.04
1	key1	row2	item1	col2	0.44	0.07
2	key1	row0	item1	col0	0.77	0.01
3	key0	row4	item0	col2	0.15	0.59
4	key1	row0	item2	col1	0.81	0.64
5	key1	row2	item2	col4	0.13	0.88
6	key2	row4	item1	col3	0.88	0.39
7	key1	row4	item1	col1	0.10	0.07
8	key1	row0	item2	col4	0.65	0.02
9	key1	row2	item0	col2	0.35	0.61
10	key2	row0	item2	col1	0.40	0.85
11	key2	row4	item1	col2	0.64	0.25
12	key0	row2	item2	col3	0.50	0.44
13	key0	row4	item1	col4	0.24	0.46
14	key1	row3	item2	col3	0.28	0.11
15	key0	row3	item1	col1	0.31	0.23
16	key0	row0	item2	col3	0.86	0.01
17	key0	row4	item0	col3	0.64	0.21
18	key2	row2	item2	col0	0.13	0.45
19	key0	row2	item0	col4	0.37	0.70

In [294]:
```
d = data = {'A': {0: 1, 1: 1, 2: 1, 3: 2, 4: 2, 5: 3, 6: 5},
 'B': {0: 'a', 1: 'b', 2: 'c', 3: 'a', 4: 'b', 5: 'a', 6: 'c'}}
df = pd.DataFrame(d)
df
```

Out[294]:

	A	B
0	1	a
1	1	b
2	1	c
3	2	a
4	2	b
5	3	a
6	5	c

Python Pandas Error tokenizing data

data = pd.read_csv('file1.csv', error_bad_lines=False)

Remap values in pandas column with a dict

In [295]:
```
df = pd.DataFrame({'col2': {0: 'a', 1: 2, 2: np.nan}, 'col1': {0: 'w', 1: 1,
df
```

Out[295]:

	col2	col1
0	a	w
1	2	1
2	NaN	2

In [296]:
```
di = {1: "A", 2: "B"}
df.replace({"col1": di})
```

Out[296]:

	col2	col1
0	a	w
1	2	A
2	NaN	B

Pandas read_csv low_memory and dtype options

dashboard_df = pd.read_csv(p_file, sep=',', error_bad_lines=False, index_col=False, dtype='unicode')

df = pd.read_csv('somefile.csv', low_memory=False)

Pandas - How to flatten a hierarchical index in columns

In [297]:
```
df = pd.DataFrame({'col2': {0: 'a', 1: 2, 2: np.nan}, 'col1': {0: 'w', 1: 1,
df
```

Out[297]:

	col2	col1
0	a	w
1	2	1
2	NaN	2

In [298]:
```
df.columns.get_level_values(0)
```

Out[298]:
```
Index(['col2', 'col1'], dtype='object')
```

In [299]:
```
[''.join(col).strip() for col in df.columns.values]
```

Out[299]:
```
['col2', 'col1']
```

In [300]:
```
df.columns.map(''.join).str.strip()
```

Out[300]:
```
Index(['col2', 'col1'], dtype='object')
```

How do I create test and train samples from one dataframe with pandas?

In [301]:
```
df = pd.DataFrame(np.random.randn(100, 2))
df
```

Out[301]:

	0	1
0	1.485629	0.328995
1	1.502822	-1.112604
2	-0.417898	2.226009
3	-0.530721	1.176287
4	-0.305795	0.163245
5	-0.320613	-1.900125
6	0.095495	-0.990709
7	-1.356761	0.666960
8	-1.757701	1.351598
9	0.395862	0.916287
10	0.587947	-0.779254

In [302]:
```
msk = np.random.rand(len(df)) < 0.8
```

In [303]:
```
train = df[msk]
test = df[~msk]
```

In [304]:
```
print(len(train))
print(len(test))
print(len(msk))
```

```
84
16
100
```

In [305]:
```
from sklearn.model_selection import train_test_split

train, test = train_test_split(df, test_size=0.2)
print(df.shape[0])
print(train.shape[0])
print(test.shape[0])
```

```
100
80
20
```

In [306]:
```
train = df.sample(frac=0.8,random_state=200) #random state is a seed value
test = df.drop(train.index)
print(df.shape[0])
print(train.shape[0])
print(test.shape[0])
```

```
100
80
20
```

Selecting/excluding sets of columns in pandas [duplicate]

In [307]:
```
df = pd.DataFrame(np.random.randn(100, 4), columns=list('ABCD'))
df
```

Out[307]:

	A	B	C	D
0	1.735389	-0.443606	0.106114	-0.271115
1	-1.449310	-2.522765	1.164901	-0.734788
2	0.739176	0.623386	0.024132	1.374205
3	1.238995	-1.309697	0.189792	0.413783
4	0.229596	-0.570862	-1.592307	0.613245
5	1.987903	0.278494	1.991259	1.046395
6	-1.424090	2.273811	0.372069	1.140957
7	-0.549538	-0.569854	-1.423475	0.415129
8	1.778309	-0.070244	-1.132087	0.928247
9	-0.131141	0.591417	-0.460964	-0.123961
10	-0.175413	-1.725004	0.470138	0.738519

In [308]:
```
df.drop(df.columns[[1, 2]], axis=1)
```

Out[308]:

	A	D
0	1.735389	-0.271115
1	-1.449310	-0.734788
2	0.739176	1.374205
3	1.238995	0.413783
4	0.229596	0.613245
5	1.987903	1.046395
6	-1.424090	1.140957
7	-0.549538	0.415129
8	1.778309	0.928247
9	-0.131141	-0.123961
10	-0.175413	0.738519

In [309]:
```
import numpy as np
import pandas as pd

# Create a dataframe with columns A,B,C and D
df = pd.DataFrame(np.random.randn(100, 4), columns=list('ABCD'))

# include the columns you want
df[df.columns[df.columns.isin(['A', 'B'])]]


```

Out[309]:

	A	B
0	-0.505594	-0.050953
1	-1.114797	0.834338
2	0.084959	-0.169517
3	0.459387	-0.475764
4	-0.723593	-1.928901
5	-0.391302	-0.715555
6	0.989001	0.416455
7	-3.115551	1.012482
8	-0.345568	-0.291469
9	-0.882309	-1.385349
10	0.331217	0.685817

In [310]:
```
# or more simply include columns:
df[['A', 'B']]
```

Out[310]:

	A	B
0	-0.505594	-0.050953
1	-1.114797	0.834338
2	0.084959	-0.169517
3	0.459387	-0.475764
4	-0.723593	-1.928901
5	-0.391302	-0.715555
6	0.989001	0.416455
7	-3.115551	1.012482
8	-0.345568	-0.291469
9	-0.882309	-1.385349
10	0.331217	0.685817

In [311]:
```
# exclude columns you don't want
df[df.columns[~df.columns.isin(['C','D'])]]
```

Out[311]:

	A	B
0	-0.505594	-0.050953
1	-1.114797	0.834338
2	0.084959	-0.169517
3	0.459387	-0.475764
4	-0.723593	-1.928901
5	-0.391302	-0.715555
6	0.989001	0.416455
7	-3.115551	1.012482
8	-0.345568	-0.291469
9	-0.882309	-1.385349
10	0.331217	0.685817

In [312]:
```
# or even simpler since 0.24
# with the caveat that it reorders columns alphabetically
df[df.columns.difference(['C', 'D'])]
```

Out[312]:

	A	B
0	-0.505594	-0.050953
1	-1.114797	0.834338
2	0.084959	-0.169517
3	0.459387	-0.475764
4	-0.723593	-1.928901
5	-0.391302	-0.715555
6	0.989001	0.416455
7	-3.115551	1.012482
8	-0.345568	-0.291469
9	-0.882309	-1.385349
10	0.331217	0.685817

How to check whether a pandas DataFrame is empty?

In [313]:
```
df
```

Out[313]:

	A	B	C	D
0	-0.505594	-0.050953	0.710132	0.401762
1	-1.114797	0.834338	0.089076	1.639470
2	0.084959	-0.169517	1.393063	-0.901674
3	0.459387	-0.475764	1.653671	0.874155
4	-0.723593	-1.928901	-0.593842	0.831155
5	-0.391302	-0.715555	-0.376849	0.253037
6	0.989001	0.416455	0.804828	-0.857278
7	-3.115551	1.012482	1.762230	-2.711010
8	-0.345568	-0.291469	0.643225	0.861524
9	-0.882309	-1.385349	-0.397711	1.019197
10	0.331217	0.685817	1.548529	1.091730

In [314]:
```
df.empty
```

Out[314]:
```
False
```

In [315]:
```
1 len(df) == 0
```

Out[315]: False

In [316]:
```
1 len(df.index) == 0
```

Out[316]: False

Pandas - Get first row value of a given column

In [317]:
```
1 df
```

Out[317]:

	A	B	C	D
0	-0.505594	-0.050953	0.710132	0.401762
1	-1.114797	0.834338	0.089076	1.639470
2	0.084959	-0.169517	1.393063	-0.901674
3	0.459387	-0.475764	1.653671	0.874155
4	-0.723593	-1.928901	-0.593842	0.831155
5	-0.391302	-0.715555	-0.376849	0.253037
6	0.989001	0.416455	0.804828	-0.857278
7	-3.115551	1.012482	1.762230	-2.711010
8	-0.345568	-0.291469	0.643225	0.861524
9	-0.882309	-1.385349	-0.397711	1.019197
10	0.331217	0.685817	1.548529	1.091730

In [318]:
```
1 df['A'].iloc[0]
```

Out[318]: -0.5055941579202301

How to store a dataframe using Pandas

In [319]:
```
df
```

Out[319]:

	A	B	C	D
0	-0.505594	-0.050953	0.710132	0.401762
1	-1.114797	0.834338	0.089076	1.639470
2	0.084959	-0.169517	1.393063	-0.901674
3	0.459387	-0.475764	1.653671	0.874155
4	-0.723593	-1.928901	-0.593842	0.831155
5	-0.391302	-0.715555	-0.376849	0.253037
6	0.989001	0.416455	0.804828	-0.857278
7	-3.115551	1.012482	1.762230	-2.711010
8	-0.345568	-0.291469	0.643225	0.861524
9	-0.882309	-1.385349	-0.397711	1.019197
10	0.331217	0.685817	1.548529	1.091730

In [320]:
```
df.to_pickle("file_name.pkl")  # where to save it, usually as a .pkl
```

In [321]:
```
# Then you can load it back using:
df = pd.read_pickle("file_name.pkl")
df
```

Out[321]:

	A	B	C	D
0	-0.505594	-0.050953	0.710132	0.401762
1	-1.114797	0.834338	0.089076	1.639470
2	0.084959	-0.169517	1.393063	-0.901674
3	0.459387	-0.475764	1.653671	0.874155
4	-0.723593	-1.928901	-0.593842	0.831155
5	-0.391302	-0.715555	-0.376849	0.253037
6	0.989001	0.416455	0.804828	-0.857278
7	-3.115551	1.012482	1.762230	-2.711010
8	-0.345568	-0.291469	0.643225	0.861524
9	-0.882309	-1.385349	-0.397711	1.019197
10	0.331217	0.685817	1.548529	1.091730

In [322]:
```
# Another popular choice is to use HDF5 (pytables) which offers very fast ac
import pandas as pd
store = pd.HDFStore('store.h5')

store['df'] = df  # save it
store['df']  # load it
```

Out[322]:

	A	B	C	D
0	-0.505594	-0.050953	0.710132	0.401762
1	-1.114797	0.834338	0.089076	1.639470
2	0.084959	-0.169517	1.393063	-0.901674
3	0.459387	-0.475764	1.653671	0.874155
4	-0.723593	-1.928901	-0.593842	0.831155
5	-0.391302	-0.715555	-0.376849	0.253037
6	0.989001	0.416455	0.804828	-0.857278
7	-3.115551	1.012482	1.762230	-2.711010
8	-0.345568	-0.291469	0.643225	0.861524
9	-0.882309	-1.385349	-0.397711	1.019197
10	0.331217	0.685817	1.548529	1.091730

pickle: original ASCII data format

cPickle, a C library

pickle-p2: uses the newer binary format

json: standardlib json library

json-no-index: like json, but without index

msgpack: binary JSON alternative

CSV

hdfstore: HDF5 storage format

Pandas conditional creation of a series/dataframe column

In [323]:
```
df
```

Out[323]:

	A	B	C	D
0	-0.505594	-0.050953	0.710132	0.401762
1	-1.114797	0.834338	0.089076	1.639470
2	0.084959	-0.169517	1.393063	-0.901674
3	0.459387	-0.475764	1.653671	0.874155
4	-0.723593	-1.928901	-0.593842	0.831155
5	-0.391302	-0.715555	-0.376849	0.253037
6	0.989001	0.416455	0.804828	-0.857278
7	-3.115551	1.012482	1.762230	-2.711010
8	-0.345568	-0.291469	0.643225	0.861524
9	-0.882309	-1.385349	-0.397711	1.019197
10	0.331217	0.685817	1.548529	1.091730

In [324]:
```
import pandas as pd
import numpy as np

df = pd.DataFrame({'Type':list('ABBC'), 'Set':list('ZZXY')})
df
```

Out[324]:

	Type	Set
0	A	Z
1	B	Z
2	B	X
3	C	Y

In [325]:
```
df['color'] = np.where(df['Set']=='Z', 'green', 'red')
print(df)
```

```
  Type Set  color
0    A   Z  green
1    B   Z  green
2    B   X    red
3    C   Y    red
```

In [326]:
```
conditions = [
    (df['Set'] == 'Z') & (df['Type'] == 'A'),
    (df['Set'] == 'Z') & (df['Type'] == 'B'),
    (df['Type'] == 'B')]
choices = ['yellow', 'blue', 'purple']
df['color'] = np.select(conditions, choices, default='black')
print(df)
```

```
  Type Set   color
0    A   Z  yellow
1    B   Z    blue
2    B   X  purple
3    C   Y   black
```

In [327]:
```
df['color'] = ['red' if x == 'Z' else 'green' for x in df['Set']]
df
```

Out[327]:

	Type	Set	color
0	A	Z	red
1	B	Z	red
2	B	X	green
3	C	Y	green

In [328]:
```
import pandas as pd
import numpy as np

df = pd.DataFrame({'Type':list('ABBC'), 'Set':list('ZZXY')})
%timeit df['color'] = ['red' if x == 'Z' else 'green' for x in df['Set']]
%timeit df['color'] = np.where(df['Set']=='Z', 'green', 'red')
%timeit df['color'] = df.Set.map( lambda x: 'red' if x == 'Z' else 'green')
```

```
174 µs ± 12 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
343 µs ± 3.95 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
295 µs ± 4.03 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

In [329]:
```
def set_color(row):
    if row["Set"] == "Z":
        return "red"
    elif row["Type"] == "C":
        return "blue"
    else:
        return "green"

df = df.assign(color=df.apply(set_color, axis=1))

print(df)
```

```
  Type Set  color
0    A   Z    red
1    B   Z    red
2    B   X  green
3    C   Y   blue
```

pandas: filter rows of DataFrame with operator chaining

In [330]:
```
df = pd.DataFrame(np.random.randn(30, 3), columns=['a','b','c'])
df
```

Out[330]:

	a	b	c
0	-0.901585	0.754662	0.929104
1	2.884808	-1.303840	0.126876
2	0.411225	0.139640	0.239281
3	-1.200964	-3.356600	-0.666590
4	-1.971630	0.390280	-0.841268
5	0.200902	-0.911127	-2.114653
6	-0.578113	-0.536352	1.609485
7	1.507590	0.875801	-1.424558
8	-0.158199	0.374577	0.011430
9	1.435116	1.379447	0.603002
10	-0.039314	1.533693	1.592624
11	2.114491	0.519400	-1.614442
12	1.189883	0.853031	2.044493
13	0.189909	0.552629	0.365152
14	-0.949710	0.470691	-0.211660
15	-0.509894	-0.702146	-0.302421
16	1.582452	1.542417	1.496355
17	1.382349	0.295923	1.742018
18	-0.548545	0.207893	0.201407
19	-0.118010	-0.488037	0.758100
20	-0.699042	-0.856712	-0.887693
21	-2.648161	-1.632171	-1.214346
22	-0.561941	-0.241723	0.362589
23	-0.423723	1.612785	-0.605000
24	0.456249	1.225753	0.706861
25	0.135537	-0.491462	-1.283687
26	-0.558219	3.014946	0.489582
27	-0.093903	0.833013	0.102892
28	-0.006770	-0.799717	-0.555594
29	0.567297	-0.582519	0.210959

In [331]:
```
df.query('a > 0').query('0 < b < 0.5')
```

Out[331]:

	a	b	c
2	0.411225	0.139640	0.239281
17	1.382349	0.295923	1.742018

In [332]:
```
df.query('a > 0 and 0 < b < 0.5')
```

Out[332]:

	a	b	c
2	0.411225	0.139640	0.239281
17	1.382349	0.295923	1.742018

Count the frequency that a value occurs in a dataframe column

In [333]:
```
df = pd.DataFrame({'col':list('abssbab')})
df
```

Out[333]:

	col
0	a
1	b
2	s
3	s
4	b
5	a
6	b

In [334]:
```
df['col'].value_counts()
```

Out[334]:
```
b    3
s    2
a    2
Name: col, dtype: int64
```

In [335]:
```
df.groupby('col').count()
```

Out[335]:

col
a
b
s

In [336]:
```
df['freq']=df.groupby('col')['col'].transform('count')
df
```

Out[336]:

	col	freq
0	a	2
1	b	3
2	s	2
3	s	2
4	b	3
5	a	2
6	b	3

In [337]:
```
df = pd.DataFrame({'col':list('abssbab')})
df.apply(pd.value_counts)
```

Out[337]:

	col
b	3
s	2
a	2

In [338]:
```
df.apply(pd.value_counts).fillna(0)
```

Out[338]:

	col
b	3
s	2
a	2

How to select all columns, except one column in pandas?

In [339]:
```
import pandas
import numpy as np
df = pd.DataFrame(np.random.rand(4,4), columns = list('abcd'))
df
```

Out[339]:

	a	b	c	d
0	0.093626	0.339925	0.528676	0.977409
1	0.013812	0.627821	0.419035	0.627533
2	0.731452	0.075437	0.819021	0.293523
3	0.453252	0.613537	0.255077	0.974010

In [340]:
```
df.loc[:, df.columns != 'b']

```

Out[340]:

	a	c	d
0	0.093626	0.528676	0.977409
1	0.013812	0.419035	0.627533
2	0.731452	0.819021	0.293523
3	0.453252	0.255077	0.974010

In [341]:
```
df.drop('b', axis=1)
```

Out[341]:

	a	c	d
0	0.093626	0.528676	0.977409
1	0.013812	0.419035	0.627533
2	0.731452	0.819021	0.293523
3	0.453252	0.255077	0.974010

In [342]:
```
df = pd.DataFrame(np.random.rand(4,4), columns = list('abcd'))
df[df.columns.difference(['b'])]
```

Out[342]:

	a	c	d
0	0.977530	0.406738	0.473588
1	0.779719	0.751301	0.997553
2	0.315639	0.315252	0.233635
3	0.023480	0.328951	0.538361

In [343]:
```
df.loc[:, ~df.columns.isin(['a', 'b'])]

```

Out[343]:

	c	d
0	0.406738	0.473588
1	0.751301	0.997553
2	0.315252	0.233635
3	0.328951	0.538361

How to group dataframe rows into list in pandas groupby

In [344]:
```
df = pd.DataFrame( {'a':['A','A','B','B','B','C'], 'b':[1,2,5,5,4,6]})
df
```

Out[344]:

	a	b
0	A	1
1	A	2
2	B	5
3	B	5
4	B	4
5	C	6

In [345]:
```
df.groupby('a')['b'].apply(list)
```

Out[345]:
```
a
A       [1, 2]
B    [5, 5, 4]
C          [6]
Name: b, dtype: object
```

Convert Python dict into a dataframe

In [346]:

```
d = {u'2012-06-08': 388,
 u'2012-06-09': 388,
 u'2012-06-10': 388,
 u'2012-06-11': 389,
 u'2012-06-12': 389,
 u'2012-06-13': 389,
 u'2012-06-14': 389,
 u'2012-06-15': 389,
 u'2012-06-16': 389,
 u'2012-06-17': 389,
 u'2012-06-18': 390,
 u'2012-06-19': 390,
 u'2012-06-20': 390,
 u'2012-06-21': 390,
 u'2012-06-22': 390,
 u'2012-06-23': 390,
 u'2012-06-24': 390,
 u'2012-06-25': 391,
 u'2012-06-26': 391,
 u'2012-06-27': 391,
 u'2012-06-28': 391,
 u'2012-06-29': 391,
 u'2012-06-30': 391,
 u'2012-07-01': 391,
 u'2012-07-02': 392,
 u'2012-07-03': 392,
 u'2012-07-04': 392,
 u'2012-07-05': 392,
 u'2012-07-06': 392}
```

In [347]:
```
df = pd.DataFrame(d.items())
df
```

Out[347]:

	0	1
0	2012-06-08	388
1	2012-06-09	388
2	2012-06-10	388
3	2012-06-11	389
4	2012-06-12	389
5	2012-06-13	389
6	2012-06-14	389
7	2012-06-15	389
8	2012-06-16	389
9	2012-06-17	389
10	2012-06-18	390
11	2012-06-19	390
12	2012-06-20	390
13	2012-06-21	390
14	2012-06-22	390
15	2012-06-23	390
16	2012-06-24	390
17	2012-06-25	391
18	2012-06-26	391
19	2012-06-27	391
20	2012-06-28	391
21	2012-06-29	391
22	2012-06-30	391
23	2012-07-01	391
24	2012-07-02	392
25	2012-07-03	392
26	2012-07-04	392
27	2012-07-05	392
28	2012-07-06	392

In [348]:
```
1 pd.DataFrame(d.items(), columns=['Date', 'DateValue'])
```

Out[348]:

	Date	DateValue
0	2012-06-08	388
1	2012-06-09	388
2	2012-06-10	388
3	2012-06-11	389
4	2012-06-12	389
5	2012-06-13	389
6	2012-06-14	389
7	2012-06-15	389
8	2012-06-16	389
9	2012-06-17	389
10	2012-06-18	390
11	2012-06-19	390
12	2012-06-20	390
13	2012-06-21	390
14	2012-06-22	390
15	2012-06-23	390
16	2012-06-24	390
17	2012-06-25	391
18	2012-06-26	391
19	2012-06-27	391
20	2012-06-28	391
21	2012-06-29	391
22	2012-06-30	391
23	2012-07-01	391
24	2012-07-02	392
25	2012-07-03	392
26	2012-07-04	392
27	2012-07-05	392
28	2012-07-06	392

How to check if a column exists in Pandas

In [349]:
```
df = pd.DataFrame( {'a':['A','A','B','B','B','C'], 'b':[1,2,5,5,4,6]})
df
```

Out[349]:

	a	b
0	A	1
1	A	2
2	B	5
3	B	5
4	B	4
5	C	6

In [350]:
```
'a' in df
```

Out[350]: True

In [351]:
```
'a' in df.columns
```

Out[351]: True

In [352]:
```
all([item in df.columns for item in ['a','b']])
```

Out[352]: True

In [353]:
```
all([item in df.columns for item in ['a','c']])
```

Out[353]: False

In [354]:
```
{'a', 'b'}.issubset(df.columns)
```

Out[354]: True

In [355]:
```
{'a', 'c'}.issubset(df.columns)
```

Out[355]: False

What is the most efficient way to loop through dataframes with pandas? ¶

In [356]:
```
'''for index, row in df.iterrows():

    # do some logic here'''
```

Out[356]: 'for index, row in df.iterrows():\n\n   # do some logic here'

Get list from pandas dataframe column or row?

In [357]:
```
import pandas as pd

data_dict = {'one': pd.Series([1, 2, 3], index=['a', 'b', 'c']),
             'two': pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}

df = pd.DataFrame(data_dict)
print(f"DataFrame:\n{df}\n")
print(f"column types:\n{df.dtypes}")

col_one_list = df['one'].tolist()

col_one_arr = df['one'].to_numpy()

print(f"\ncol_one_list:\n{col_one_list}\ntype:{type(col_one_list)}")
print(f"\ncol_one_arr:\n{col_one_arr}\ntype:{type(col_one_arr)}")

```

```
DataFrame:
   one  two
a  1.0    1
b  2.0    2
c  3.0    3
d  NaN    4

column types:
one    float64
two      int64
dtype: object

col_one_list:
[1.0, 2.0, 3.0, nan]
type:<class 'list'>

col_one_arr:
[ 1.  2.  3. nan]
type:<class 'numpy.ndarray'>
```

In [ ]:
```

```