# NumPy & Pandas

GopalKumar Katariya , 210133062005

# Outline

- What is NumPy ?
- How is NumPy used in data science ?
- Is NumPy used in AI ?
- NumPy arrays
- NumPy Indexing and Selection
- NumPy Operations
- Series
- DataFrames
- Groupby
- Merging, Joining, and Concatenating
- Pandas Operations

# What is NumPy ?

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

# How is NumPy used in data science ?

NumPy provides an efficient interface to store and operate on dense data buffers. In some ways, NumPy arrays are like Python's built-in list type, but NumPy arrays provide much more efficient storage and data operations as the arrays grow larger in size.

# Is NumPy used in AI?

NumPy library is an important foundational tool for studying Machine Learning. Many of its functions are very useful for performing any mathematical or scientific calculation. As it is known that mathematics is the foundation of machine learning, most of the mathematical tasks can be performed using NumPy.

# NumPy arrays

- We can create an array by directly converting a list or list of lists.

```
my_list = [1,2,3]
np.array(my_list)
```

```
my_matrix = [[1,2,3],[4,5,6],[7,8,9]]
np.array(my_matrix)
```

# NumPy arrays

- There are lots of built-in ways to generate Arrays .

```
np.arange(0,10)
```

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

```
np.arange(0,11,2)
```

```
array([0, 2, 4, 6, 8, 10])
```

```
np.zeros(2,2)
```

```
array([[0., 0.,],
       [0., 0.,],)
```

# NumPy arrays

- Return evenly spaced numbers over a specified interval.

```
np.linspace(0,10,3)
```

```
array([0., 5., 10.])
```

# NumPy arrays

- Numpy also has lots of ways to create random number arrays:

```
np.random.rand(2,2)
```

```
array([0.23433564, 0.35467895])
```

```
np.random.randn(2,2)
```

```
array([-0.24378564, 0.15437695])
```

# NumPy arrays

- Returns an array containing the same data with a new shape.

```
arr = np.arange(25)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
17, 18, 19, 20, 21, 22, 23, 24])
```

```
arr.reshape(5,5)
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14],
       [15, 16, 17, 18, 19],
       [20, 21, 22, 23, 24]])
```

# NumPy Indexing and Selection

- Using bracket

```
arr[8]
```

```
8
```

```
arr[1:5]
```

```
array([1,  2,  3,  4])
```

# NumPy Indexing and Selection

- Broadcasting

```
arr[0:5]=100
```

```
array([100, 100, 100, 100, 100,   5,   6,   7,   8,   9,  10])
```

# NumPy Indexing and Selection

- Data is not copied, it's a view of the original array .

```
arr_copy = arr.copy()
```

# NumPy Indexing and Selection

- Indexing a 2D array

```
arr_2d[1]
```

```
array([20, 25, 30])
```

```
arr_2d[1][0]
```

```
20
```

```
arr_2d[:2,1:]
```

```
array([[10, 15],
       [25, 30]])
```

# NumPy Operations

- You can easily perform array with array arithmetic, or scalar with array arithmetic.

```
arr + arr
```

```
arr * arr
```

```
arr - arr
```

```
arr / arr  # division by zero replaced with nan
```

# NumPy Operations

- Universal Array Functions.

```
np.sqrt(arr)
```

```
np.exp(arr)
```

```
np.max(arr)
```

```
np.sin(arr)
```

# Series

- You can convert a list, numpy array, or dictionary to a Series.

```
labels = [‘a','b','c']
my_list = [10,20,30]
arr = np.array([10,20,30])
d = {'a':10,'b':20,'c':30}
```

# Series

- Using Lists

```
pd.Series(data = my_list)
```

```
0    10
1    20
2    30
dtype: int64
```

```
pd.Series(data = my_list, index = labels)
```

```
a    10
b    20
c    30
dtype: int64
```

# Series

- NumPy Arrays

```
pd.Series(arr)
```

```
0    10
1    20
2    30
dtype: int64
```

```
pd.Series(arr, labeles)
```

```
a    10
b    20
c    30
dtype: int64
```

# Series

- Dictionary

```
pd.Series(d)
```

```
a    10
b    20
c    30
dtype: int64
```

# DataFrames

- Dictionary

```
df = pd.DataFrame(randn(5,4),
             index='A B C D E’.split(),
             columns='W X Y Z’.split())
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

# DataFrames

- Selection and Indexing

```
df[‘W’]
```

```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

```
type(df[‘W’])
```

pandas.core.series.Series

# DataFrames

- Creating a new column

```
df['new'] = df['W'] + df['Y']
```

```
          W         X         Y         Z       new
A  2.706850  0.628133  0.907969  0.503826  3.614819
B  0.651118 -0.319318 -0.848077  0.605965 -0.196959
C -2.018168  0.740122  0.528813 -0.589001 -1.489355
D  0.188695 -0.758872 -0.933237  0.955057 -0.744542
E  0.190794  1.978757  2.605967  0.683509  2.796762
```

```
df.drop(‘new’,axis=1) #remove column
```

# DataFrames

- Selecting Rows

```
df.loc['A']
```

```
W   -2.018168
X    0.740122
Y    0.528813
Z   -0.589001
Name: C, dtype: float64
```

```
df.drop(‘new’,axis=1) #remove column
```

# DataFrames

- Conditional Selection

```
df > 0
```

```
      W      X      Y      Z
A  True   True   True   True
B  True  False  False   True
C False   True   True  False
D  True  False  False   True
E  True   True   True   True
```

# DataFrames

- Conditional Selection

```
df[df[‘W'] > 0]
```

```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

```
df[df['W']>0]['Y']
```

```
A     0.907969
B    -0.848077
D    -0.933237
E     2.605967t64
```

# Groupby

- Create dataframe

```
data = {'Company':['GOOG','GOOG','MSFT','MSFT','FB','FB'],
       'Person':['Sam','Charlie','Amy','Vanessa','Carl','Sarah'],
       ‘Sales':[200,120,340,124,243,350]}
df = pd.DataFrame(data)
```

| | Company | Person | Sales |
|---|---|---|---|
| 0 | GOOG | Sam | 200 |
| 1 | GOOG | Charlie | 120 |
| 2 | MSFT | Amy | 340 |
| 3 | MSFT | Vanessa | 124 |
| 4 | FB | Carl | 243 |
| 5 | FB | Sarah | 350 |

# Groupby

- Save this object as a new variable and find mean .

```
by_comp = df.groupby(“Company”)

by_comp.mean()
```

| | Sales |
|---|---|
| Company | |
| FB | 296.5 |
| GOOG | 160.0 |
| MSFT | 232.0 |

# Groupby

- Describe

```
by_comp.describe()
```

| | Sales | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | count | mean | std | min | 25% | 50% | 75% | max |
| Company | | | | | | | | |
| FB | 2.0 | 296.5 | 75.660426 | 243.0 | 269.75 | 296.5 | 323.25 | 350.0 |
| GOOG | 2.0 | 160.0 | 56.568542 | 120.0 | 140.00 | 160.0 | 180.00 | 200.0 |
| MSFT | 2.0 | 232.0 | 152.735065 | 124.0 | 178.00 | 232.0 | 286.00 | 340.0 |

# Merging, Joining, and Concatenating

- DataFrames

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', ‘A3’],
                    'B': ['B0', 'B1', 'B2', ‘B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', ‘D3']},
                    index=[0, 1, 2, 3])
Df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', ‘A7’],
                    'B': ['B4', 'B5', 'B6', ‘B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', ‘D7']},
                    index=[4, 5, 6, 7])
```

# Merging, Joining, and Concatenating

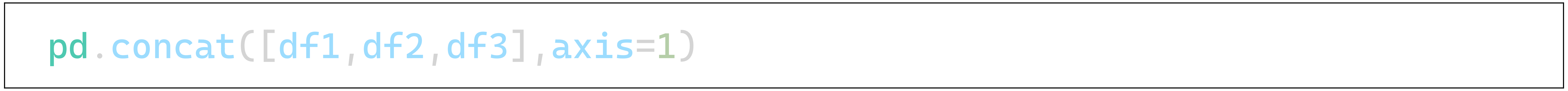
- Concatenation

```
pd.concat([df1,df2,df3])
```

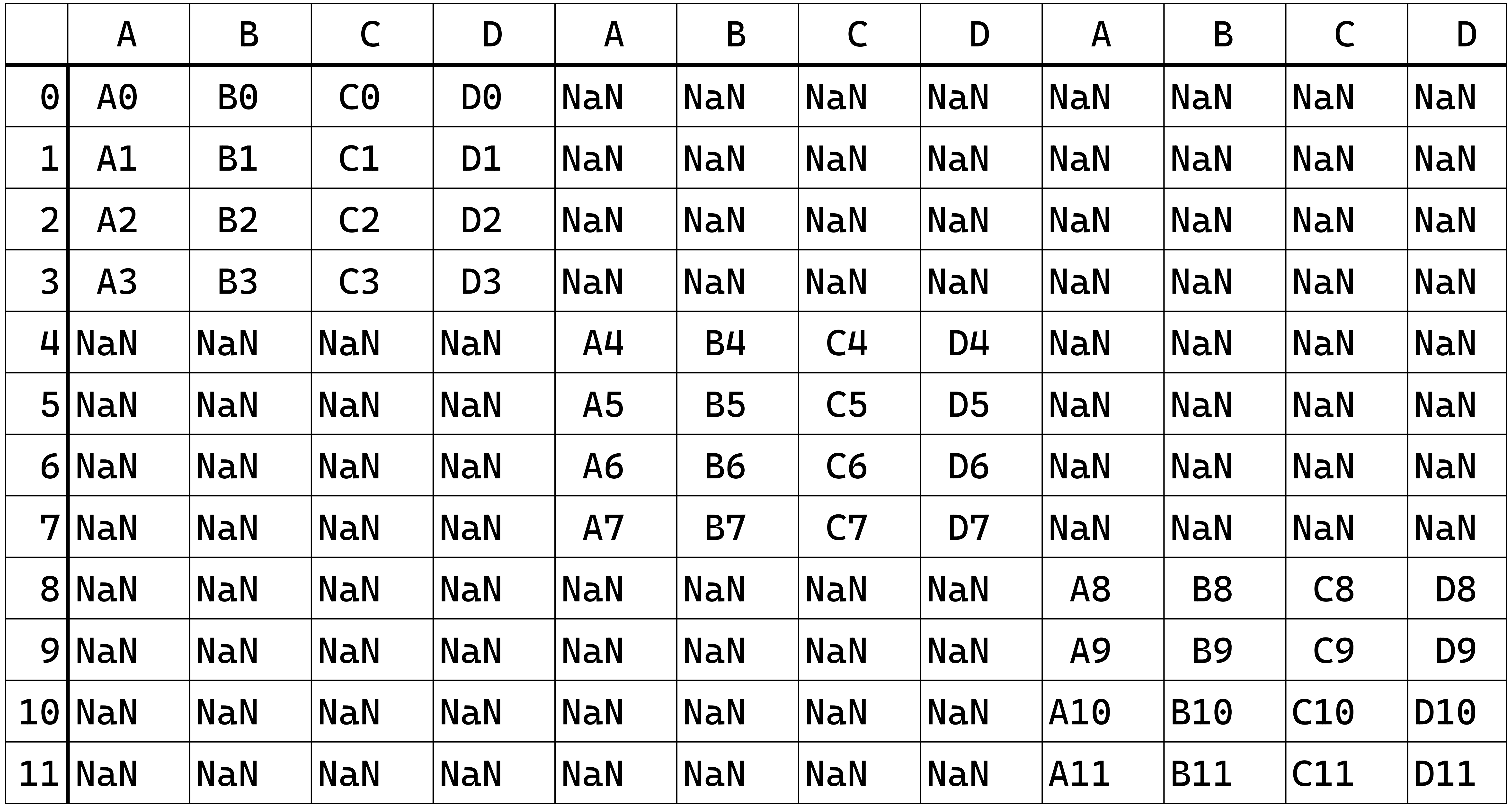
| | A | B | C | D |
|---|---|---|---|---|
| 0 | A0 | B0 | C0 | D0 |
| 1 | A1 | B1 | C1 | D1 |
| 2 | A2 | B2 | C2 | D2 |
| 3 | A3 | B3 | C3 | D3 |
| 4 | A4 | B4 | C4 | D4 |
| 5 | A5 | B5 | C5 | D5 |
| 6 | A6 | B6 | C6 | D6 |
| 7 | A7 | B7 | C7 | D7 |
| 8 | A8 | B8 | C8 | D8 |
| 9 | A9 | B9 | C9 | D9 |
| 10 | A10 | B10 | C10 | D10 |
| 11 | A11 | B11 | C11 | D11 |

# Merging, Joining, and Concatenating

- Concatenation



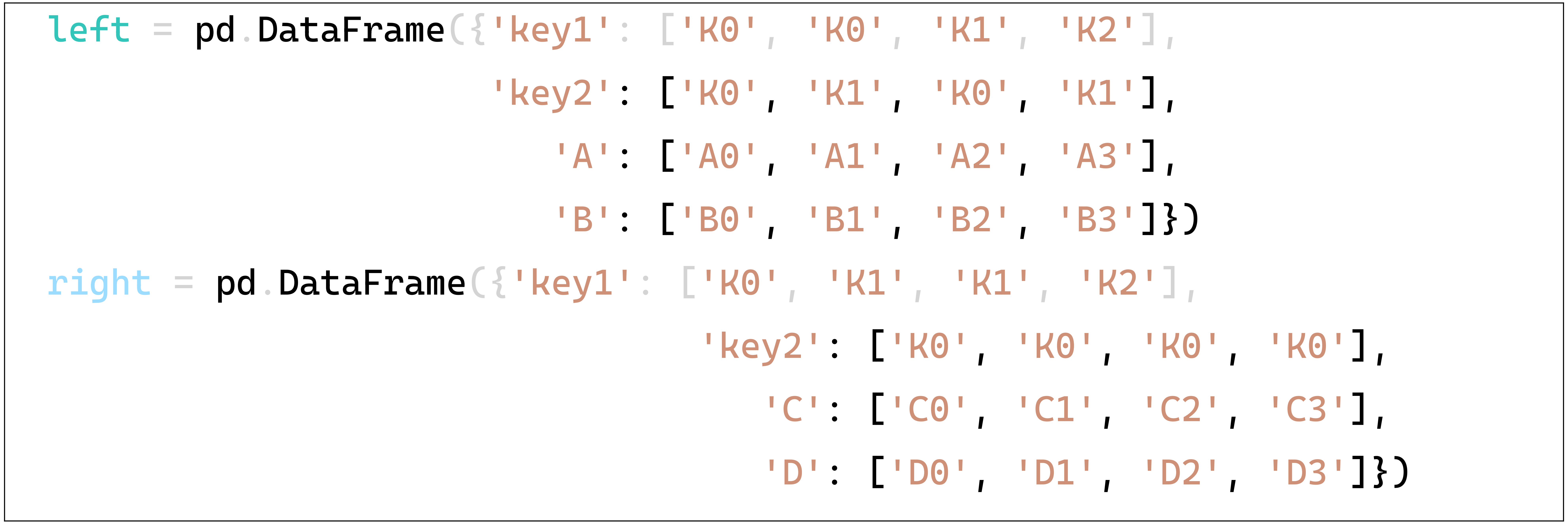
```
pd.concat([df1,df2,df3],axis=1)
```



| | A | B | C | D | A | B | C | D | A | B | C | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | A0 | B0 | C0 | D0 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 1 | A1 | B1 | C1 | D1 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 2 | A2 | B2 | C2 | D2 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 3 | A3 | B3 | C3 | D3 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN |
| 4 | NaN | NaN | NaN | NaN | A4 | B4 | C4 | D4 | NaN | NaN | NaN | NaN |
| 5 | NaN | NaN | NaN | NaN | A5 | B5 | C5 | D5 | NaN | NaN | NaN | NaN |
| 6 | NaN | NaN | NaN | NaN | A6 | B6 | C6 | D6 | NaN | NaN | NaN | NaN |
| 7 | NaN | NaN | NaN | NaN | A7 | B7 | C7 | D7 | NaN | NaN | NaN | NaN |
| 8 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | A8 | B8 | C8 | D8 |
| 9 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | A9 | B9 | C9 | D9 |
| 10 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | A10 | B10 | C10 | D10 |
| 11 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | A11 | B11 | C11 | D11 |

# Merging, Joining, and Concatenating

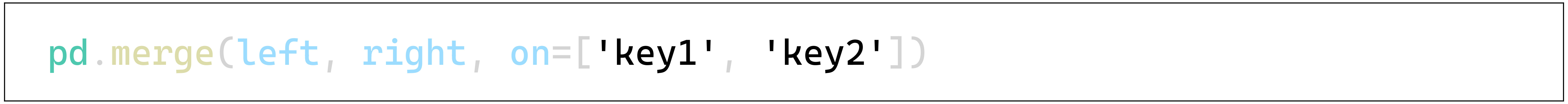
- Merging



```
left = pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'],
                     'key2': ['K0', 'K1', 'K0', 'K1'],
                        'A': ['A0', 'A1', 'A2', 'A3'],
                        'B': ['B0', 'B1', 'B2', 'B3']})
right = pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'],
                               'key2': ['K0', 'K0', 'K0', 'K0'],
                                  'C': ['C0', 'C1', 'C2', 'C3'],
                                  'D': ['D0', 'D1', 'D2', 'D3']})
```

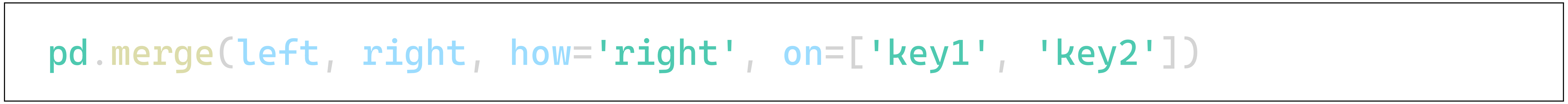
# Merging, Joining, and Concatenating

- Merging



```
pd.merge(left, right, on=['key1', 'key2'])
```

| | key1 | key2 | A | B | C | D |
|---|---|---|---|---|---|---|
| 0 | K0 | K0 | A0 | B0 | C0 | D0 |
| 1 | K1 | K0 | A2 | B2 | C1 | D1 |
| 2 | K1 | K0 | A2 | B2 | C2 | D2 |



```
pd.merge(left, right, how='right', on=['key1', 'key2'])
```

| | key1 | key2 | A | B | C | D |
|---|---|---|---|---|---|---|
| 0 | K0 | K0 | A0 | B0 | C0 | D0 |
| 1 | K1 | K0 | A2 | B2 | C1 | D1 |
| 2 | K1 | K0 | A2 | B2 | C2 | D2 |
| 3 | K2 | K0 | NaN | NaN | C3 | D3 |

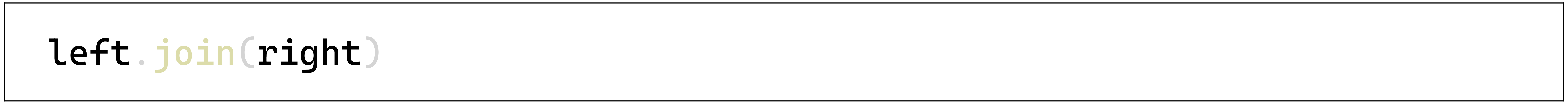
# Merging, Joining, and Concatenating

- Joining

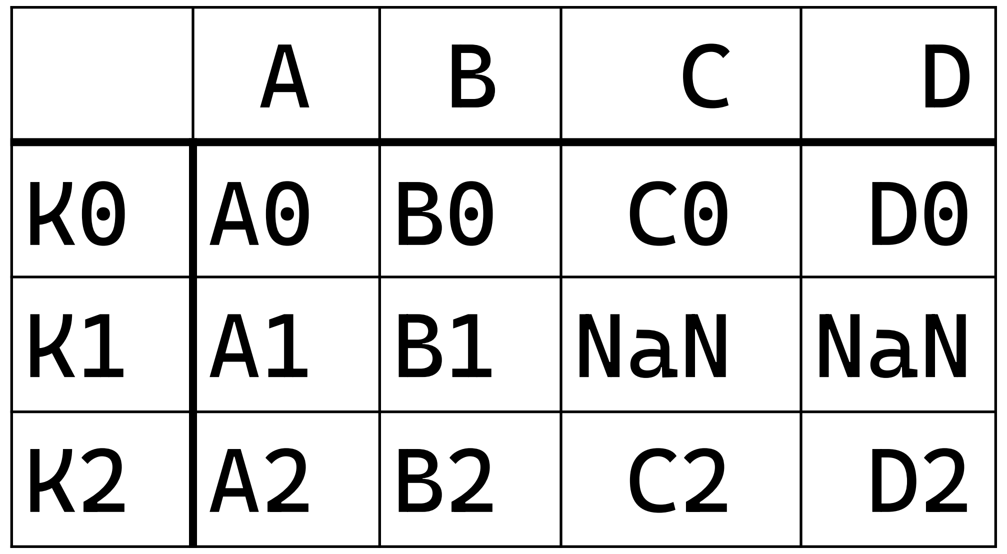
```
left = pd.DataFrame({'A': ['A0', 'A1', 'A2'],
                     'B': ['B0', 'B1', 'B2']},
                      index=['K0', 'K1', 'K2'])

right = pd.DataFrame({'C': ['C0', 'C2', 'C3'],
                    'D': ['D0', 'D2', 'D3']},
                      index=['K0', 'K2', 'K3'])
```

# Merging, Joining, and Concatenating

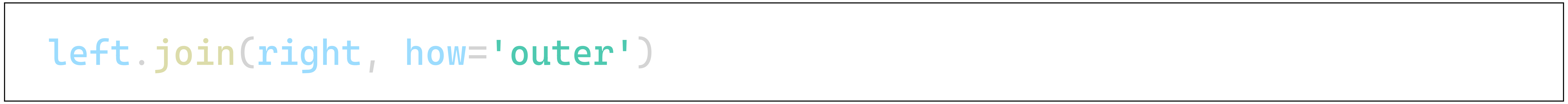
- Merging



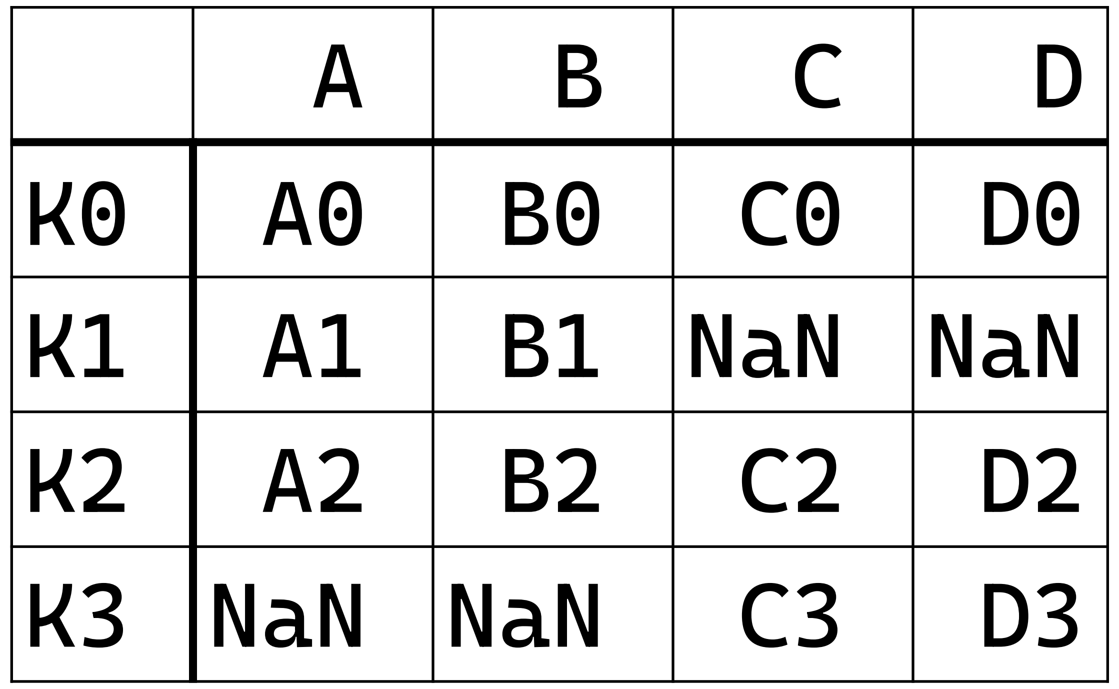
```
left.join(right)
```



|  | A | B | C | D |
|---|---|---|---|---|
| K0 | A0 | B0 | C0 | D0 |
| K1 | A1 | B1 | NaN | NaN |
| K2 | A2 | B2 | C2 | D2 |



```
left.join(right, how='outer')
```



|  | A | B | C | D |
|---|---|---|---|---|
| K0 | A0 | B0 | C0 | D0 |
| K1 | A1 | B1 | NaN | NaN |
| K2 | A2 | B2 | C2 | D2 |
| K3 | NaN | NaN | C3 | D3 |

# Pandas Operations

- Create dataframe.

```
df = pd.DataFrame({‘col1’:[1,2,3,4],
                   ‘col2':[444,555,666,444],
                   ‘col3’:[‘abc’,'def','ghi','xyz']})
df.head()
```

| | col1 | col2 | col3 |
|---|---|---|---|
| 0 | 1 | 444 | abc |
| 1 | 2 | 555 | def |
| 2 | 3 | 666 | ghi |
| 3 | 4 | 444 | xyz |

# Pandas Operations

- There are lots of operations in pandas.

```
df['col2'].unique()
```

```
array([444, 555, 666])
```

```
df['col2'].nunique()
```

```
3
```

```
df['col2'].value_counts()
```

```
444    2
555    1
666    1
Name: col2, dtype: int64
```

# Pandas Operations

- Selecting Data

```
df[(df[‘col1'] > 2) & (df[‘col2'] == 444)]
```

```
f   col1  col2 col3
3     4   444  xyz
```

# Pandas Operations

- Applying Functions

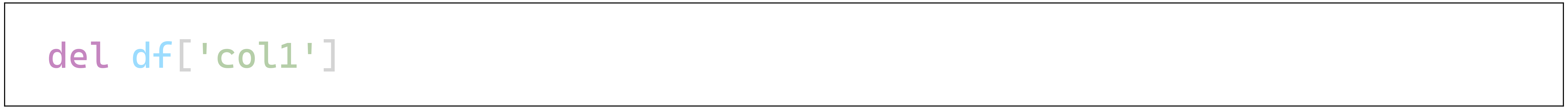
```
def times2(x):
    return x * 2
```

```
df['col1'].apply(times2)
```

```
0    2
1    4
2    6
3    8
Name: col1, dtype: int64
```

# Pandas Operations

- Permanently Removing a Column



```
del df['col1']
```

# Pandas Operations

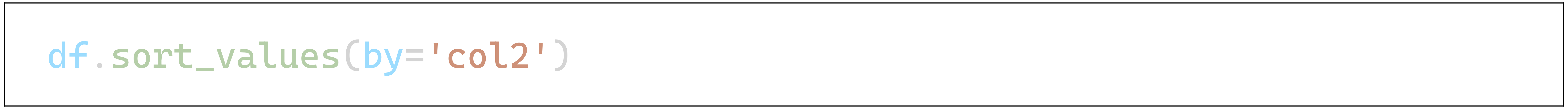
- Get column and index names

```
df.columns
```

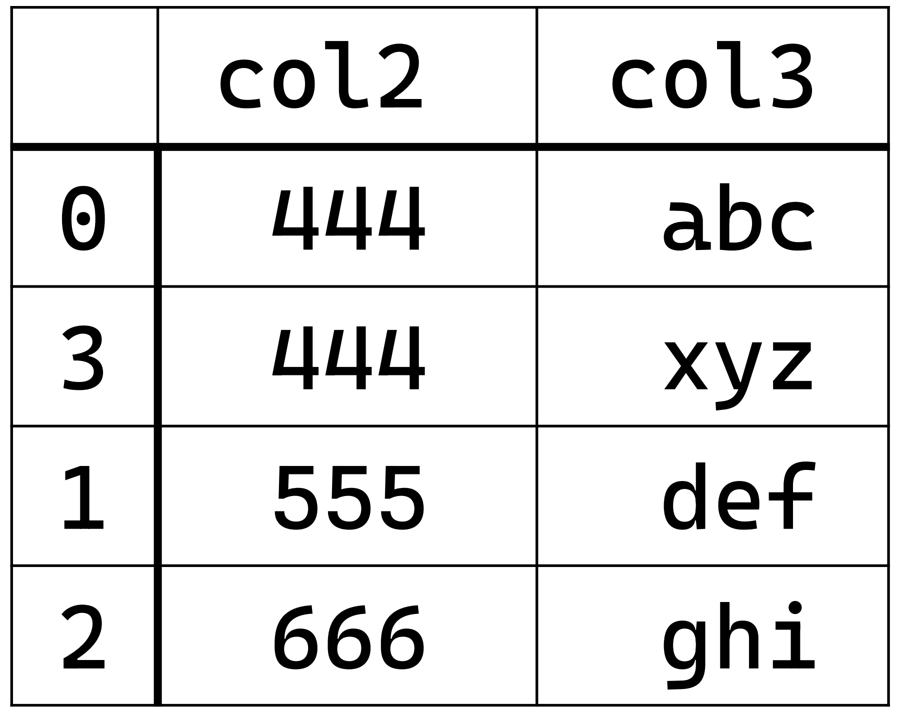
```
df.index
```

# Pandas Operations

- Sorting and Ordering a DataFrame



```
df.sort_values(by='col2')
```



| | col2 | col3 |
|---|---|---|
| 0 | 444 | abc |
| 3 | 444 | xyz |
| 1 | 555 | def |
| 2 | 666 | ghi |

# Pandas Operations

- Find Null Values or Check for Null Values

```
df.isnull()
```

|   | col2 | col3 |
|---|---|---|
| 0 | False | FALSE |
| 1 | False | FALSE |
| 2 | False | FALSE |
| 3 | False | FALSE |

```
df.dropna()
```

|   | col2 | col3 |
|---|---|---|
| 0 | 444 | abc |
| 1 | 555 | def |
| 2 | 666 | ghi |
| 3 | 444 | xyz |

# Thank You