Lecture 5-1

Pandas

Week 5 Monday

Miles Chen, PhD

Pandas

NumPy creates ndarrays that must contain values that are of the same data type.

Pandas creates dataframes. Each column in a dataframe is an ndarray. This allows us to have traditional tables of data where each column can be a different data type.

Important References:

https://pandas.pydata.org/pandas-docs/stable/reference/series.html

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

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

The basic data structure in pandas is the *series*. You can construct it in a similar fashion to making a numpy array.

The command to make a Series object is

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

data could be a list
index is the equivalent of a row_name

the `index` argument is optional

The basic data structure in pandas is the *series*. You can construct it in a similar fashion to making a numpy array.

The command to make a Series object is

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

the `index` argument is optional

In [2]:
```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
print(data)
print(type(data))
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
<class 'pandas.core.series.Series'>
```

The basic data structure in pandas is the *series*. You can construct it in a similar fashion to making a numpy array.

The command to make a Series object is

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

the `index` argument is optional

In [2]:
```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
print(data)
print(type(data))
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
<class 'pandas.core.series.Series'>
```

In [3]:
```
data
```

Out[3]:
```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

The basic data structure in pandas is the *series*. You can construct it in a similar fashion to making a numpy array.

The command to make a Series object is

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

the `index` argument is optional

In [2]:
```
data = pd.Series([0.25, 0.5, 0.75, 1.0])
print(data)
print(type(data))
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
<class 'pandas.core.series.Series'>
```

In [3]:
```
data
```

Out[3]:
```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
```

The series is printed out in a table form. The type is a Pandas Series

In [4]:
```
print(data.values)
```

```
[0.25 0.5  0.75 1.  ]
```

In [5]:
```
print(type(data.values))
```

```
<class 'numpy.ndarray'>
```

The values attribute of the series is a numpy array.

when you pair the values with an index, the numpy array is now a series

In [4]:
```
print(data.values)
```

```
[0.25 0.5  0.75 1.  ]
```

In [5]:
```
print(type(data.values))
```

```
<class 'numpy.ndarray'>
```

The values attribute of the series is a numpy array.

In [6]:
```
print(data.index)
```

```
RangeIndex(start=0, stop=4, step=1)
```

In [7]:
```
print(type(data.index))  # the row names are known as the index
```

```
<class 'pandas.core.indexes.range.RangeIndex'>
```

You can subset a pandas series like other python objects

In [8]:
```
print(data[1])
```

```
0.5
```

In [9]:
```
print(type(data[1]))  # when you select only one value, it simplifies the object
```

```
<class 'numpy.float64'>
```

Can subset a series like any other python iterable object

You can subset a pandas series like other python objects

In [8]:
```
print(data[1])
```

```
0.5
```

In [9]:
```
print(type(data[1]))  # when you select only one value, it simplifies the object
```

```
<class 'numpy.float64'>
```

In [10]:
```
print(data[1:3])
```

since its returning two numbers, it maintains the series structure

```
1    0.50
2    0.75
dtype: float64
```

In [11]:
```
print(type(data[1:3]))  # slicing / selecting multiple values returns a series
```

```
<class 'pandas.core.series.Series'>
```

You can subset a pandas series like other python objects

In [8]:
```
print(data[1])
```

```
0.5
```

In [9]:
```
print(type(data[1]))  # when you select only one value, it simplifies the object
```

```
<class 'numpy.float64'>
```

In [10]:
```
print(data[1:3])
```

```
1    0.50
2    0.75
dtype: float64
```

In [11]:
```
print(type(data[1:3]))  # slicing / selecting multiple values returns a series
```

```
<class 'pandas.core.series.Series'>
```

In [12]:
```
print(data[np.array([1, 0, 1, 2])])  # You can also do fancy indexing by subsetting w/a numpy array
```

```
1    0.50
0    0.25
1    0.50
2    0.75
dtype: float64
```

In [13]:
```
# Pandas uses a 0-based index by default. You may also specify the index values
data = pd.Series([0.25, 0.5, 0.75, 1.0],
                 index = ['a', 'b', 'c', 'd'])
print(data)
```

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
```

In [14]:
```
data.values
```

Out[14]:
```
array([0.25, 0.5 , 0.75, 1.  ])
```

In [15]:
```
data.index
```

Out[15]:
```
Index(['a', 'b', 'c', 'd'], dtype='object')
```

In [16]:
```
data[1]  # subset with index position
```

Out[16]:
```
0.5
```

In [16]:
```
data[1]  # subset with index position
```

Out[16]:
```
0.5
```

In [17]:
```
data["a"]  # subset with index names
```

Out[17]:
```
0.25
```

In [16]:
```
data[1]  # subset with index position
```

Out[16]:
```
0.5
```

In [17]:
```
data["a"]  # subset with index names
```

Out[17]:
```
0.25
```

In [18]:
```
data[0:2]  # slicing behavior is unchanged
```

Out[18]:
```
a    0.25
b    0.50
dtype: float64
```

In [16]:
```
data[1]  # subset with index position
```

Out[16]:
```
0.5
```

In [17]:
```
data["a"]  # subset with index names
```

Out[17]:
```
0.25
```

In [18]:
```
data[0:2]  # slicing behavior is unchanged
```

Out[18]:
```
a    0.25
b    0.50
dtype: float64
```

In [19]:
```
data["a":"c"] # slicing using index names includes the last value
```

Out[19]:
```
a    0.25
b    0.50
c    0.75
dtype: float64
```

In [20]:

```
# creating a series from a python dictionary
# remember, dictionary construction uses curly braces {}
samp_dict = {'Tony Stark': "Robert Downey Jr.",
             'Steve Rogers': "Chris Evans",
             'Natasha Romanoff': "Scarlett Johansson",
             'Bruce Banner': "Mark Ruffalo",
             'Thor': "Chris Hemsworth",
             'Clint Barton': "Jeremy Renner"}
samp_series = pd.Series(samp_dict)
samp_series
```

Out[20]:

```
Tony Stark           Robert Downey Jr.
Steve Rogers               Chris Evans
Natasha Romanoff    Scarlett Johansson
Bruce Banner              Mark Ruffalo
Thor                   Chris Hemsworth
Clint Barton             Jeremy Renner
dtype: object
```

In [20]:

```
# creating a series from a python dictionary
# remember, dictionary construction uses curly braces {}
samp_dict = {'Tony Stark': "Robert Downey Jr.",
             'Steve Rogers': "Chris Evans",
             'Natasha Romanoff': "Scarlett Johansson",
             'Bruce Banner': "Mark Ruffalo",
             'Thor': "Chris Hemsworth",
             'Clint Barton': "Jeremy Renner"}
samp_series = pd.Series(samp_dict)
samp_series
```

Out[20]:

```
Tony Stark           Robert Downey Jr.
Steve Rogers               Chris Evans
Natasha Romanoff    Scarlett Johansson
Bruce Banner              Mark Ruffalo
Thor                   Chris Hemsworth
Clint Barton             Jeremy Renner
dtype: object
```

In [21]:

```
print(samp_series.index) # dtype = object is for strings but allows mixed data types.
```

```
Index(['Tony Stark', 'Steve Rogers', 'Natasha Romanoff', 'Bruce Banner',
       'Thor', 'Clint Barton'],
      dtype='object')
```

In [20]:
```
# creating a series from a python dictionary
# remember, dictionary construction uses curly braces {}
samp_dict = {'Tony Stark': "Robert Downey Jr.",
             'Steve Rogers': "Chris Evans",
             'Natasha Romanoff': "Scarlett Johansson",
             'Bruce Banner': "Mark Ruffalo",
             'Thor': "Chris Hemsworth",
             'Clint Barton': "Jeremy Renner"}
samp_series = pd.Series(samp_dict)
samp_series
```

Out[20]:
```
Tony Stark           Robert Downey Jr.
Steve Rogers               Chris Evans
Natasha Romanoff    Scarlett Johansson
Bruce Banner              Mark Ruffalo
Thor                   Chris Hemsworth
Clint Barton             Jeremy Renner
dtype: object
```

In [21]:
```
print(samp_series.index) # dtype = object is for strings but allows mixed data types.
```

```
Index(['Tony Stark', 'Steve Rogers', 'Natasha Romanoff', 'Bruce Banner',
       'Thor', 'Clint Barton'],
      dtype='object')
```

In [22]:
```
samp_series.values
```

Out[22]:
```
array(['Robert Downey Jr.', 'Chris Evans', 'Scarlett Johansson',
       'Mark Ruffalo', 'Chris Hemsworth', 'Jeremy Renner'], dtype=object)
```

In [23]:

```
# ages during the First Avengers film (2012)
age_dict = {'Thor': 1493,
            'Steve Rogers': 104,
            'Natasha Romanoff': 28,
            'Clint Barton': 41,
            'Tony Stark': 42,
            'Bruce Banner': 42}  # note that the dictionary order is not same here
ages = pd.Series(age_dict)
print(ages)
```

```
Thor                1493
Steve Rogers         104
Natasha Romanoff      28
Clint Barton          41
Tony Stark            42
Bruce Banner          42
dtype: int64
```

In [24]:

```
# ages during the First Avengers film (2012)
hero_dict = {'Thor': np.NaN,
            'Steve Rogers': 'Captain America',
            'Natasha Romanoff': 'Black Widow',
            'Clint Barton': 'Hawkeye',
            'Tony Stark': 'Iron Man',
            'Bruce Banner': 'Hulk'}
hero_names = pd.Series(hero_dict)
print(hero_names)
```

```
Thor                          NaN
Steve Rogers        Captain America
Natasha Romanoff        Black Widow
Clint Barton                Hawkeye
Tony Stark                Iron Man
```

```
Bruce Banner                  Hulk
dtype: object
```

Creating a DataFrame

There are multiple ways of creating a DataFrame in Pandas. The next few slides show a few.

Creating a DataFrame

There are multiple ways of creating a DataFrame in Pandas. The next few slides show a few.

We can create a dataframe by providing a dictionary of series objects. The dictionary key becomes the column name. The dictionary values become values. The keys within the dictionaries become the index.

In [25]:
```
avengers = pd.DataFrame({'actor': samp_series,
                         'hero name': hero_names,
                         'age': ages})

# the DataFrame will match the indices and sort them
print(avengers)
```

```
                            actor        hero name   age
Bruce Banner          Mark Ruffalo             Hulk    42
Clint Barton         Jeremy Renner           Hawkeye    41
Natasha Romanoff  Scarlett Johansson       Black Widow    28
Steve Rogers            Chris Evans  Captain America   104
Thor                Chris Hemsworth               NaN  1493
Tony Stark        Robert Downey Jr.          Iron Man    42
```

Default behavior of pd.DataFrame is to alphabatize the index

Creating a DataFrame

There are multiple ways of creating a DataFrame in Pandas. The next few slides show a few.

We can create a dataframe by providing a dictionary of series objects. The dictionary key becomes the column name. The dictionary values become values. The keys within the dictionaries become the index.

In [25]:
```
avengers = pd.DataFrame({'actor': samp_series,
                         'hero name': hero_names,
                         'age': ages})

# the DataFrame will match the indices and sort them
print(avengers)
```

```
                              actor        hero name   age
Bruce Banner            Mark Ruffalo             Hulk    42
Clint Barton          Jeremy Renner           Hawkeye    41
Natasha Romanoff  Scarlett Johansson       Black Widow    28
Steve Rogers            Chris Evans  Captain America   104
Thor                 Chris Hemsworth               NaN  1493
Tony Stark         Robert Downey Jr.          Iron Man    42
```

In [26]:
```
print(type(avengers))  # this is a DataFrame object
```

```
<class 'pandas.core.frame.DataFrame'>
```

The data is a list of dictionaries. Each dictionary needs to have the same set of keys, otherwise, NaNs will appear.

The data is a list of dictionaries. Each dictionary needs to have the same set of keys, otherwise, NaNs will appear.

In [27]:
```
data = [{'a': 0, 'b': 0},
        {'a': 1, 'b': 2},
        {'a': 2, 'b': 5}]
data
```

Out[27]:
```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 5}]
```

The data is a list of dictionaries. Each dictionary needs to have the same set of keys, otherwise, NaNs will appear.

In [27]:
```
data = [{'a': 0, 'b': 0},
        {'a': 1, 'b': 2},
        {'a': 2, 'b': 5}]
data
```

Out[27]:
```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 5}]
```

In [28]:
```
print(pd.DataFrame(data, index = [1, 2, 3]))
```

```
   a  b
1  0  0
2  1  2
3  2  5
```

The data is a list of dictionaries. Each dictionary needs to have the same set of keys, otherwise, NaNs will appear.

In [27]:
```
data = [{'a': 0, 'b': 0},
        {'a': 1, 'b': 2},
        {'a': 2, 'b': 5}]
data
```

Out[27]:
```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 5}]
```

In [28]:
```
print(pd.DataFrame(data, index = [1, 2, 3]))
```

```
   a  b
1  0  0
2  1  2
3  2  5
```

In [29]:
```
data2 = [{'a': 0, 'b': 0},
         {'a': 1, 'b': 2},
         {'a': 2, 'c': 5}] # mismatch of keys. NAs will appear
data2
```

Out[29]:
```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'c': 5}]
```

The data is a list of dictionaries. Each dictionary needs to have the same set of keys, otherwise, NaNs will appear.

In [27]:
```
data = [{'a': 0, 'b': 0},
        {'a': 1, 'b': 2},
        {'a': 2, 'b': 5}]
data
```

Out[27]:
```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'b': 5}]
```

In [28]:
```
print(pd.DataFrame(data, index = [1, 2, 3]))
```

```
   a  b
1  0  0
2  1  2
3  2  5
```

In [29]:
```
data2 = [{'a': 0, 'b': 0},
         {'a': 1, 'b': 2},
         {'a': 2, 'c': 5}] # mismatch of keys. NAs will appear
data2
```

Out[29]:
```
[{'a': 0, 'b': 0}, {'a': 1, 'b': 2}, {'a': 2, 'c': 5}]
```

In [30]:
```
pd.DataFrame(data2)  # if the index argument is not supplied, it defaults to integer index start at
```

Out[30]:

	a	b	c
0	0	0.0	NaN
1	1	2.0	NaN
2	2	NaN	5.0

You can convert a dictionary to a DataFrame. The keys form column names, and the values are lists/arrays of values. The arrays need to be of the same length.

You can convert a dictionary to a DataFrame. The keys form column names, and the values are lists/arrays of values. The arrays need to be of the same length.

In [31]:
```
data3 = {'a': [1, 2, 3],
         'b': ['x','y','z']}
data3
```

Out[31]:
```
{'a': [1, 2, 3], 'b': ['x', 'y', 'z']}
```

You can convert a dictionary to a DataFrame. The keys form column names, and the values are lists/arrays of values. The arrays need to be of the same length.

In [31]:
```
data3 = {'a': [1, 2, 3],
         'b': ['x','y','z']}
data3
```

Out[31]:
```
{'a': [1, 2, 3], 'b': ['x', 'y', 'z']}
```

In [32]:
```
pd.DataFrame(data3)
```

Out[32]:

	a	b
0	1	x
1	2	y
2	3	z

You can convert a dictionary to a DataFrame. The keys form column names, and the values are lists/arrays of values. The arrays need to be of the same length.

In [31]:
```
data3 = {'a': [1, 2, 3],
         'b': ['x','y','z']}
data3
```

Out[31]:
```
{'a': [1, 2, 3], 'b': ['x', 'y', 'z']}
```

In [32]:
```
pd.DataFrame(data3)
```

Out[32]:

	a	b
0	1	x
1	2	y
2	3	z

In [33]:
```
data4 = {'a': [1, 2, 3, 4],
         'b': ['x','y','z']} # arrays are not of the same length
pd.DataFrame(data4)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-33-32abcf74ba0a> in <module>
      1 data4 = {'a': [1, 2, 3, 4],
      2          'b': ['x','y','z']} # arrays are not of the same length
----> 3 pd.DataFrame(data4)

~\anaconda3\lib\site-packages\pandas\core\frame.py in __init__(self, data, index,
columns, dtype, copy)
    527
    528         elif isinstance(data, dict):
--> 529             mgr = init_dict(data, index, columns, dtype=dtype)
```

```
    530         elif isinstance(data, ma.MaskedArray):
    531             import numpy.ma.mrecords as mrecords

~\anaconda3\lib\site-packages\pandas\core\internals\construction.py in init_dict
(data, index, columns, dtype)
    285             arr if not is_datetime64tz_dtype(arr) else arr.copy() for arr
in arrays
    286         ]
--> 287     return arrays_to_mgr(arrays, data_names, index, columns, dtype=dtype)
    288
    289

~\anaconda3\lib\site-packages\pandas\core\internals\construction.py in arrays_to_
mgr(arrays, arr_names, index, columns, dtype, verify_integrity)
     78         # figure out the index, if necessary
     79         if index is None:
---> 80             index = extract_index(arrays)
     81         else:
     82             index = ensure_index(index)

~\anaconda3\lib\site-packages\pandas\core\internals\construction.py in extract_in
dex(data)
    399             lengths = list(set(raw_lengths))
    400             if len(lengths) > 1:
--> 401                 raise ValueError("arrays must all be same length")
    402
    403             if have_dicts:

ValueError: arrays must all be same length
```

Turn a 2D Numpy array (matrix) into a DataFrame by adding column names and optionally index values.

Turn a 2D Numpy array (matrix) into a DataFrame by adding column names and optionally index values.

In [34]:
```
data = np.random.randint(10, size = 10).reshape((5,2))
print(data)
```

```
[[0 8]
 [7 1]
 [9 2]
 [5 8]
 [0 7]]
```

Turn a 2D Numpy array (matrix) into a DataFrame by adding column names and optionally index values.

In [34]:
```
data = np.random.randint(10, size = 10).reshape((5,2))
print(data)
```

```
[[0 8]
 [7 1]
 [9 2]
 [5 8]
 [0 7]]
```

In [35]:
```
print(pd.DataFrame(data, columns = ["x","y"], index = ['a','b','c','d','e']))
```

```
   x  y
a  0  8
b  7  1
c  9  2
d  5  8
e  0  7
```

Subsetting the DataFrame

In a DataFrame, the `.columns` attribute show the column names and the `.index` attribute show the row names.

Subsetting the DataFrame

In a DataFrame, the `.columns` attribute show the column names and the `.index` attribute show the row names.

In [36]:
```
print(avengers)
```

```
                              actor        hero name   age
Bruce Banner            Mark Ruffalo             Hulk    42
Clint Barton          Jeremy Renner           Hawkeye    41
Natasha Romanoff  Scarlett Johansson      Black Widow    28
Steve Rogers             Chris Evans  Captain America   104
Thor                  Chris Hemsworth              NaN  1493
Tony Stark          Robert Downey Jr.         Iron Man    42
```

Subsetting the DataFrame

In a DataFrame, the `.columns` attribute show the column names and the `.index` attribute show the row names.

In [36]:
```
print(avengers)
```

```
                              actor        hero name   age
Bruce Banner            Mark Ruffalo             Hulk    42
Clint Barton           Jeremy Renner          Hawkeye    41
Natasha Romanoff  Scarlett Johansson      Black Widow    28
Steve Rogers             Chris Evans  Captain America   104
Thor                Chris Hemsworth              NaN  1493
Tony Stark         Robert Downey Jr.         Iron Man    42
```

In [37]:
```
print(avengers.columns)
```

```
Index(['actor', 'hero name', 'age'], dtype='object')
```

In [38]:
```
print(avengers.index)
```

```
Index(['Bruce Banner', 'Clint Barton', 'Natasha Romanoff', 'Steve Rogers',
       'Thor', 'Tony Stark'],
      dtype='object')
```

You can select a column using dot notation or with single square brackets.

You can select a column using dot notation or with single square brackets.

In [39]:
```
avengers.actor  # extracting the column
```

Out[39]:
```
Bruce Banner              Mark Ruffalo
Clint Barton             Jeremy Renner
Natasha Romanoff    Scarlett Johansson
Steve Rogers               Chris Evans
Thor                     Chris Hemsworth
Tony Stark            Robert Downey Jr.
Name: actor, dtype: object
```

You can select a column using dot notation or with single square brackets.

In [39]:
```
avengers.actor  # extracting the column
```

Out[39]:
```
Bruce Banner              Mark Ruffalo
Clint Barton             Jeremy Renner
Natasha Romanoff    Scarlett Johansson
Steve Rogers               Chris Evans
Thor                    Chris Hemsworth
Tony Stark            Robert Downey Jr.
Name: actor, dtype: object
```

In [40]:
```
avengers["hero name"] # if there's a space in the column name, you'll need to use square brackets
```

Out[40]:
```
Bruce Banner                  Hulk
Clint Barton                Hawkeye
Natasha Romanoff        Black Widow
Steve Rogers        Captain America
Thor                            NaN
Tony Stark                 Iron Man
Name: hero name, dtype: object
```

You can select a column using dot notation or with single square brackets.

In [39]:
```
avengers.actor  # extracting the column
```

Out[39]:
```
Bruce Banner              Mark Ruffalo
Clint Barton             Jeremy Renner
Natasha Romanoff    Scarlett Johansson
Steve Rogers               Chris Evans
Thor                    Chris Hemsworth
Tony Stark             Robert Downey Jr.
Name: actor, dtype: object
```

In [40]:
```
avengers["hero name"] # if there's a space in the column name, you'll need to use square brackets
```

Out[40]:
```
Bruce Banner                  Hulk
Clint Barton                Hawkeye
Natasha Romanoff        Black Widow
Steve Rogers        Captain America
Thor                            NaN
Tony Stark                 Iron Man
Name: hero name, dtype: object
```

In [41]:
```
type(avengers.actor)
```

Out[41]:
```
pandas.core.series.Series
```

The selected column is a Pandas Series and can be subset accordingly.

The selected column is a Pandas Series and can be subset accordingly.

In [42]:
```
avengers.actor[1] # 0 based indexing
```

Out[42]:
```
'Jeremy Renner'
```

The selected column is a Pandas Series and can be subset accordingly.

In [42]:
```
avengers.actor[1] # 0 based indexing
```

Out[42]:
```
'Jeremy Renner'
```

In [43]:
```
avengers.actor[avengers.age == 42]
```

This is huge

Out[43]:
```
Bruce Banner         Mark Ruffalo
Tony Stark      Robert Downey Jr.
Name: actor, dtype: object
```

The selected column is a Pandas Series and can be subset accordingly.

In [42]:
```
avengers.actor[1] # 0 based indexing
```

Out[42]:
```
'Jeremy Renner'
```

In [43]:
```
avengers.actor[avengers.age == 42]
```

Out[43]:
```
Bruce Banner         Mark Ruffalo
Tony Stark      Robert Downey Jr.
Name: actor, dtype: object
```

In [44]:
```
avengers["hero name"]['Steve Rogers']
```

Out[44]:
```
'Captain America'
```

The selected column is a Pandas Series and can be subset accordingly.

In [42]:
```
avengers.actor[1] # 0 based indexing
```

Out[42]:
```
'Jeremy Renner'
```

In [43]:
```
avengers.actor[avengers.age == 42]
```

Out[43]:
```
Bruce Banner         Mark Ruffalo
Tony Stark      Robert Downey Jr.
Name: actor, dtype: object
```

In [44]:
```
avengers["hero name"]['Steve Rogers']
```

Out[44]:
```
'Captain America'
```

In [45]:
```
avengers["hero name"]['Steve Rogers':'Tony Stark']
```

Out[45]:
```
Steve Rogers    Captain America
Thor                        NaN
Tony Stark             Iron Man
Name: hero name, dtype: object
```

`.loc`

The `.loc` attribute can be used to subset the DataFrame using the index names.

.loc

The `.loc` attribute can be used to subset the DataFrame using the index names.

In [46]:
```
avengers.loc['Thor'] # subset based on location to get a row
```

Out[46]:
```
actor        Chris Hemsworth
hero name                NaN
age                     1493
Name: Thor, dtype: object
```

In [47]:
```
print(type(avengers.loc['Thor']))
print(type(avengers.loc['Thor'].values))  # the values are of mixed type but is still a numpy array.
# this is possible because it is a structured numpy array. (covered in "Python for Data Science" cha
```

```
<class 'pandas.core.series.Series'>
<class 'numpy.ndarray'>
```

equivalent to avengers.age

In [48]:
```
print(avengers.loc[ : ,'age']) # subset based on location to get a column
```

```
Bruce Banner          42
Clint Barton          41
Natasha Romanoff      28
Steve Rogers         104
Thor                1493
Tony Stark            42
Name: age, dtype: int64
```

In [49]:
```
print(type(avengers.loc[:,'age']))  #the object is a pandas series
print(type(avengers.loc[:,'age'].values))
```

```
<class 'pandas.core.series.Series'>
<class 'numpy.ndarray'>
```

In [48]:
```
print(avengers.loc[ : ,'age']) # subset based on location to get a column
```

```
Bruce Banner          42
Clint Barton          41
Natasha Romanoff      28
Steve Rogers         104
Thor                1493
Tony Stark            42
Name: age, dtype: int64
```

In [49]:
```
print(type(avengers.loc[:,'age']))  #the object is a pandas series
print(type(avengers.loc[:,'age'].values))
```

```
<class 'pandas.core.series.Series'>
<class 'numpy.ndarray'>
```

In [50]:
```
avengers.loc['Steve Rogers','age']  # you can provide a pair of 'coordinates' to get a particular va
```

Out[50]:
```
104
```

`.iloc`

The `.iloc` attribute can be used to subset the DataFrame using the index position (zero-indexed).

`.iloc`

The `.iloc` attribute can be used to subset the DataFrame using the index position (zero-indexed).

In [51]:
```
avengers.iloc[3,] # subset based on index location
```

Out[51]:
```
actor            Chris Evans
hero name    Captain America
age                      104
Name: Steve Rogers, dtype: object
```

`.iloc`

The `.iloc` attribute can be used to subset the DataFrame using the index position (zero-indexed).

In [51]:
```
avengers.iloc[3,] # subset based on index location
```

Out[51]:
```
actor            Chris Evans
hero name    Captain America
age                      104
Name: Steve Rogers, dtype: object
```

In [52]:
```
avengers.iloc[0, 1] # pair of coordinates
```

Out[52]:
```
'Hulk'
```

Assignment with `.loc` and `.iloc`

The `.loc` and `.iloc` attributes can be used in conjunction with assignment.

Assignment with `.loc` and `.iloc`

The `.loc` and `.iloc` attributes can be used in conjunction with assignment.

In [53]:
```
# set values individually
avengers.loc['Thor', 'age'] = 1500
avengers.loc['Thor', 'hero name'] = 'Thor'
avengers
```

Out[53]:

	actor	hero name	age
Bruce Banner	Mark Ruffalo	Hulk	42
Clint Barton	Jeremy Renner	Hawkeye	41
Natasha Romanoff	Scarlett Johansson	Black Widow	28
Steve Rogers	Chris Evans	Captain America	104
Thor	Chris Hemsworth	Thor	1500
Tony Stark	Robert Downey Jr.	Iron Man	42

Assignment with `.loc` and `.iloc`

The `.loc` and `.iloc` attributes can be used in conjunction with assignment.

In [53]:
```
# set values individually
avengers.loc['Thor', 'age'] = 1500
avengers.loc['Thor', 'hero name'] = 'Thor'
avengers
```

Out[53]:

	actor	hero name	age
Bruce Banner	Mark Ruffalo	Hulk	42
Clint Barton	Jeremy Renner	Hawkeye	41
Natasha Romanoff	Scarlett Johansson	Black Widow	28
Steve Rogers	Chris Evans	Captain America	104
Thor	Chris Hemsworth	Thor	1500
Tony Stark	Robert Downey Jr.	Iron Man	42

In [54]:
```
# assign multiple values at once
avengers.loc['Thor', ['hero name', 'age']] = [np.NaN, 1493]
avengers
```

Out[54]:

	actor	hero name	age
Bruce Banner	Mark Ruffalo	Hulk	42
Clint Barton	Jeremy Renner	Hawkeye	41
Natasha Romanoff	Scarlett Johansson	Black Widow	28
Steve Rogers	Chris Evans	Captain America	104
Thor	Chris Hemsworth	NaN	1493
Tony Stark	Robert Downey Jr.	Iron Man	42

`.loc` vs `.iloc` with numeric index

The following DataFrame has a numeric index, but it starts at 1 instead of 0.

`.loc` vs `.iloc` with numeric index

The following DataFrame has a numeric index, but it starts at 1 instead of 0.

In [55]:
```
data = [{'a': 11, 'b': 2},
        {'a': 12, 'b': 4},
        {'a': 13, 'b': 6}]
df = pd.DataFrame(data, index = [1, 2, 3])
df
```

Out[55]:

	a	b
1	11	2
2	12	4
3	13	6

`.loc` vs `.iloc` with numeric index

The following DataFrame has a numeric index, but it starts at 1 instead of 0.

In [55]:
```
data = [{'a': 11, 'b': 2},
        {'a': 12, 'b': 4},
        {'a': 13, 'b': 6}]
df = pd.DataFrame(data, index = [1, 2, 3])
df
```

Out[55]:

	a	b
1	11	2
2	12	4
3	13	6

In [56]:
```
df.loc[1, :] # .loc always uses the actual index.
```

Out[56]:
```
a    11
b     2
Name: 1, dtype: int64
```

`.loc` vs `.iloc` with numeric index

The following DataFrame has a numeric index, but it starts at 1 instead of 0.

In [55]:
```
data = [{'a': 11, 'b': 2},
        {'a': 12, 'b': 4},
        {'a': 13, 'b': 6}]
df = pd.DataFrame(data, index = [1, 2, 3])
df
```

Out[55]:

	a	b
1	11	2
2	12	4
3	13	6

In [56]:
```
df.loc[1, :] # .loc always uses the actual index.
```

Out[56]:
```
a    11
b     2
Name: 1, dtype: int64
```

In [57]:
```
df.iloc[1, :] # .iloc always uses the position using a 0-based index.
```

Out[57]:
```
a    12
b     4
Name: 2, dtype: int64
```

In [58]:

```
df.iloc[3, :] # using a position that doesn't exist results in an exception.
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-58-14a2fe1b33fd> in <module>
----> 1 df.iloc[3, :] # using a position that doesn't exist results in an excepti
on.

~\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
    887                     # AttributeError for IntervalTree get_value
    888                     return self.obj._get_value(*key, takeable=self._takea
ble)
--> 889             return self._getitem_tuple(key)
    890         else:
    891             # we by definition only have the 0th axis

~\anaconda3\lib\site-packages\pandas\core\indexing.py in _getitem_tuple(self, tu
p)
   1448     def _getitem_tuple(self, tup: Tuple):
   1449
-> 1450         self._has_valid_tuple(tup)
   1451         with suppress(IndexingError):
   1452             return self._getitem_lowerdim(tup)

~\anaconda3\lib\site-packages\pandas\core\indexing.py in _has_valid_tuple(self, k
ey)
    721         for i, k in enumerate(key):
    722             try:
--> 723                 self._validate_key(k, i)
    724             except ValueError as err:
    725                 raise ValueError(

~\anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_key(self, key,
axis)
```

```
   1356                 return
   1357             elif is_integer(key):
-> 1358                 self._validate_integer(key, axis)
   1359             elif isinstance(key, tuple):
   1360                 # a tuple should already have been caught by this point

~\anaconda3\lib\site-packages\pandas\core\indexing.py in _validate_integer(self,
 key, axis)
   1442             len_axis = len(self.obj._get_axis(axis))
   1443             if key >= len_axis or key < -len_axis:
-> 1444                 raise IndexError("single positional indexer is out-of-bounds"
)
   1445
   1446     # ------------------------------------------------------------------

IndexError: single positional indexer is out-of-bounds
```

Boolean subsetting examples with `.loc`

Boolean subsetting examples with `.loc`

In [59]:

```
# select avengers whose age is less than 50 and greater than 40
# select the columns 'hero name' and 'age'
avengers.loc[ (avengers.age < 50) & (avengers.age > 40), ['hero name', 'age']]
```

Out[59]:

	hero name	age
Bruce Banner	Hulk	42
Clint Barton	Hawkeye	41
Tony Stark	Iron Man	42

Boolean subsetting examples with `.loc`

In [59]:
```
# select avengers whose age is less than 50 and greater than 40
# select the columns 'hero name' and 'age'
avengers.loc[ (avengers.age < 50) & (avengers.age > 40), ['hero name', 'age']]
```

Out[59]:

	hero name	age
Bruce Banner	Hulk	42
Clint Barton	Hawkeye	41
Tony Stark	Iron Man	42

In [60]:
```
# Use the index of the DataFrame, treat it as a string, and select rows that start with B
avengers.loc[ avengers.index.str.startswith('B'), : ]
```

Out[60]:

	actor	hero name	age
Bruce Banner	Mark Ruffalo	Hulk	42

Boolean subsetting examples with `.loc`

In [59]:
```
# select avengers whose age is less than 50 and greater than 40
# select the columns 'hero name' and 'age'
avengers.loc[ (avengers.age < 50) & (avengers.age > 40), ['hero name', 'age']]
```

Out[59]:

	hero name	age
Bruce Banner	Hulk	42
Clint Barton	Hawkeye	41
Tony Stark	Iron Man	42

In [60]:
```
# Use the index of the DataFrame, treat it as a string, and select rows that start with B
avengers.loc[ avengers.index.str.startswith('B'), : ]
```

Out[60]:

	actor	hero name	age
Bruce Banner	Mark Ruffalo	Hulk	42

In [61]:
```
# Use the index of the DataFrame, treat it as a string,
# find the character capital R. Find returns -1 if it does not find the letter
# We select rows that did not result in -1, which means it does contain a capital R
avengers.loc[ avengers.index.str.find('R') != -1, : ]
```

Out[61]:

	actor	hero name	age
Natasha Romanoff	Scarlett Johansson	Black Widow	28
Steve Rogers	Chris Evans	Captain America	104

Other commonly used DataFrame attributes

Other commonly used DataFrame attributes

In [62]:
```
avengers.T  # the transpose
```

Out[62]:

	Bruce Banner	Clint Barton	Natasha Romanoff	Steve Rogers	Thor	Tony Stark
actor	Mark Ruffalo	Jeremy Renner	Scarlett Johansson	Chris Evans	Chris Hemsworth	Robert Downey Jr.
hero name	Hulk	Hawkeye	Black Widow	Captain America	NaN	Iron Man
age	42	41	28	104	1493	42

Other commonly used DataFrame attributes

In [62]:
```
avengers.T  # the transpose
```

Out[62]:

	Bruce Banner	Clint Barton	Natasha Romanoff	Steve Rogers	Thor	Tony Stark
actor	Mark Ruffalo	Jeremy Renner	Scarlett Johansson	Chris Evans	Chris Hemsworth	Robert Downey Jr.
hero name	Hulk	Hawkeye	Black Widow	Captain America	NaN	Iron Man
age	42	41	28	104	1493	42

In [63]:
```
avengers.dtypes # the data types contained in the DataFrame
```

Out[63]:
```
actor        object
hero name    object
age           int64
dtype: object
```

Other commonly used DataFrame attributes

In [62]:
```
avengers.T  # the transpose
```

Out[62]:

	Bruce Banner	Clint Barton	Natasha Romanoff	Steve Rogers	Thor	Tony Stark
actor	Mark Ruffalo	Jeremy Renner	Scarlett Johansson	Chris Evans	Chris Hemsworth	Robert Downey Jr.
hero name	Hulk	Hawkeye	Black Widow	Captain America	NaN	Iron Man
age	42	41	28	104	1493	42

In [63]:
```
avengers.dtypes # the data types contained in the DataFrame
```

Out[63]:
```
actor        object
hero name    object
age           int64
dtype: object
```

In [64]:
```
avengers.shape # shape
```

Out[64]:
```
(6, 3)
```

Importing Data with pd.read_csv()

Importing Data with pd.read_csv()

In [65]:

```
# Titanic Dataset
url = 'https://assets.datacamp.com/production/course_1607/datasets/titanic_sub.csv'
titanic = pd.read_csv(url)
```

Importing Data with pd.read_csv()

In [65]:
```
# Titanic Dataset
url = 'https://assets.datacamp.com/production/course_1607/datasets/titanic_sub.csv'
titanic = pd.read_csv(url)
```

In [66]:
```
titanic
```

Out[66]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	male	35.0	0	0	373450	8.0500	NaN	S
...	...	...	...	...	...	...	...	...	...	...	...
886	887	0	2	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	male	32.0	0	0	370376	7.7500	NaN	Q

891 rows × 11 columns

In [67]:
```
titanic.shape
```

Out[67]:
```
(891, 11)
```

In [67]:
```
titanic.shape
```

Out[67]:
```
(891, 11)
```

In [68]:
```
titanic.columns
```

Out[68]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [67]:
```
titanic.shape
```

Out[67]:
```
(891, 11)
```

In [68]:
```
titanic.columns
```

Out[68]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

In [69]:
```
titanic.index
```

Out[69]:
```
RangeIndex(start=0, stop=891, step=1)
```

In [70]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Sex          891 non-null    object 
 4   Age          714 non-null    float64
 5   SibSp        891 non-null    int64  
 6   Parch        891 non-null    int64  
 7   Ticket       891 non-null    object 
 8   Fare         891 non-null    float64
 9   Cabin        204 non-null    object 
 10  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
```

In [71]:
```
titanic.describe()  # displays summary statistics of the numeric variables
```

Out[71]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200