Pandas Series

1405 Instructor: Ruiqing (Sam) Cao

Required Python Libraries for Today

Core

- NumPy, Pandas

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

Visualization

- Matplotlib, Seaborn

```
from matplotlib import pyplot as plt
import seaborn as sns
```

Statistical learning

- scikit-learn, SciPy, statsmodels

```
import sklearn
import scipy
import statsmodel as sm
```

NumPy & Pandas Printing Format

It's better to print only *the first few decimal digits* of large real numbers. Set the print options to keep 3 decimal digits and supress scientific notation (for NumPy arrays and Pandas DataFrames):

- **NumPy**

```
np.set_printoptions(precision=3,suppress=True)
```

- **Pandas**

```
pd.options.display.float_format = '{:.3f}'.format
```

Tabular Data (in a Pandas DataFrame)

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	'first'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

Two Pandas Data Structures

Pandas Series: a 1D labeled array that can hold any data type (similar to values of a list but with a name and indexes)

Example of a Pandas Series

Pandas Series: a 1D labeled array that can hold any data type (similar to values of a list but with a name and indexes)

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	't'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

A column series

Example of a Pandas Series

Pandas Series: a 1D labeled array that can hold any data type
(similar to values of a list but with a name and indexes)

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	'first'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

A row series

Two Pandas Data Structures

Pandas DataFrame: a 2D labeled array storing tabular data, where rows represent observations and columns represent variables

Two Pandas Data Structures

Pandas DataFrame: a 2D labeled array storing tabular data, where rows represent observations and columns represent variables

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	'first'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

A Pandas DataFrame

Anatomy of a Pandas DataFrame

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	'first'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

A **column**: presents values of a variable

Anatomy of a Pandas DataFrame

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	'first'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

A **row**: presents values of an observation

Anatomy of a Pandas DataFrame

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	'first'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

Row indexes: represent typically (but not necessarily) unique labels of observations

Anatomy of a Pandas DataFrame

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	'first'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

Column names: represent typically (but not necessarily) unique labels of variables

Anatomy of a Pandas DataFrame

	id (int)	coinflip ({0,1})	ord (str)	current (bool)
0	100	1	'zeroth'	True
1	101	0	'first'	False
2	102	0	'second'	False
3	103	1	'third'	True
4	104	1	'fourth'	True

Column data types: represent attribute (variable) data type (typically homogenous but not required)

DataFrame & Row/Column Series

- A row of a DataFrame can be represented by a Pandas Series
  - name ← a row index in the DataFrame
  - indexes ← column names in the DataFrame
  - values ← a row's values in the DataFrame

- A column of a DataFrame can be represented by a Pandas Series
  - name ← a column name in the DataFrame
  - indexes ← row indexes in the DataFrame
  - values ← a column's values in the DataFrame

Pandas Series

Anatomy of a Pandas Series

A Pandas Series s has name **coinflip**, indexes **0, 1, 2, 3, 4** and values **1, 0, 0, 1, 1**. We can retrieve these components by calling

s=

```
0    1
1    0
2    0
3    1
4    1
Name: coinflip, dtype: int64
```

`s.name` → `'coinflip'`

`s.index` → `Index([0, 1, 2, 3, 4], dtype='int64')`

`s.values` → `array([1, 0, 0, 1, 1])`

Create a Pandas Series

	pd.Series(argument)
Argument	a list, or a dictionary, or an array-like object
Returns	a Pandas Series object

```
s= pd.Series([True,True,False])
display(s)
```

output

```
0     True
1     True
2    False
dtype: bool
```

From a list

Create a Pandas Series

	pd.Series(argument)
Argument	a list, or a dictionary, or an array-like object
Returns	a Pandas Series object

```
s = pd.Series(np.ones(5),dtype=np.int32)
display(s)
```

output

```
0    1
1    1
2    1
3    1
4    1
dtype: int32
```

From a NumPy array

Create a Pandas Series

	pd.Series(argument)
Argument	a list, or a dictionary, or an array-like object
Returns	a Pandas Series object

```
s = pd.Series({'Tom':1,'Char':2,'Karen':3,'Lin':4,'Jess':5})
display(s)
```

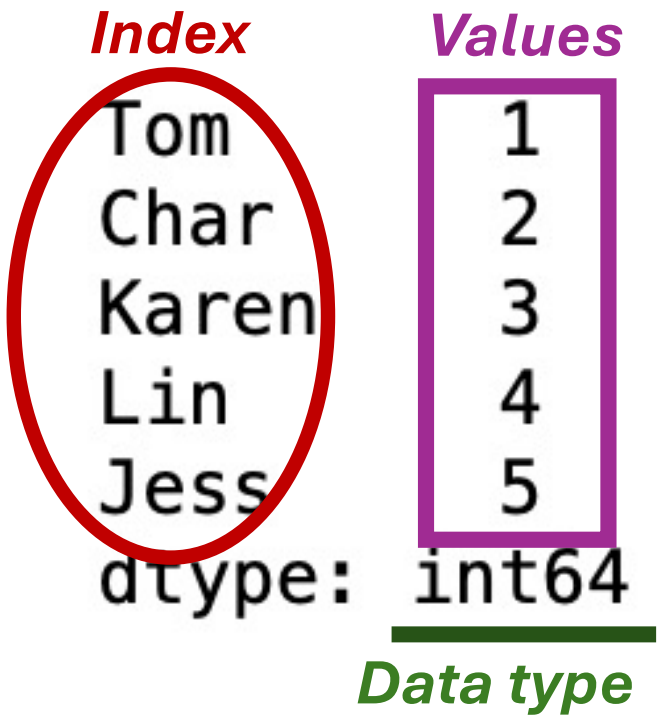
output

```
Tom      1
Char     2
Karen    3
Lin      4
Jess     5
dtype: int64
```

From a dictionary

dictionary keys → indexes of the Series
dictionary values → values of the Series

Create a Pandas Series



If no index is specified when creating a Pandas Series, it will automatically have **a default index of 0, 1, 2, …, N**, corresponding to the first N rows, starting from the top

Create a Pandas Series (With Indexes)

	pd.Series(arg,index=ind)
Arguments	arg: a list, a dictionary, or an array-like object ind: a list, or a 1D array-like object
Returns	a Pandas Series object

```
s = pd.Series(np.arange(5), index=['Tom','Char','Karen','Lin','Jess'])
display(s)
```

output

```
Tom      0
Char     1
Karen    2
Lin      3
Jess     4
dtype: int64
```

Indexes from a list

Create a Pandas Series (With Indexes)

	pd.Series(arg,index=ind)
Arguments	arg: a list, a dictionary, or an array-like object ind: a list, or a 1D array-like object
Returns	a Pandas Series object

```
s = pd.Series([None]*5, index=np.arange(5)*2)
display(s)
```

output

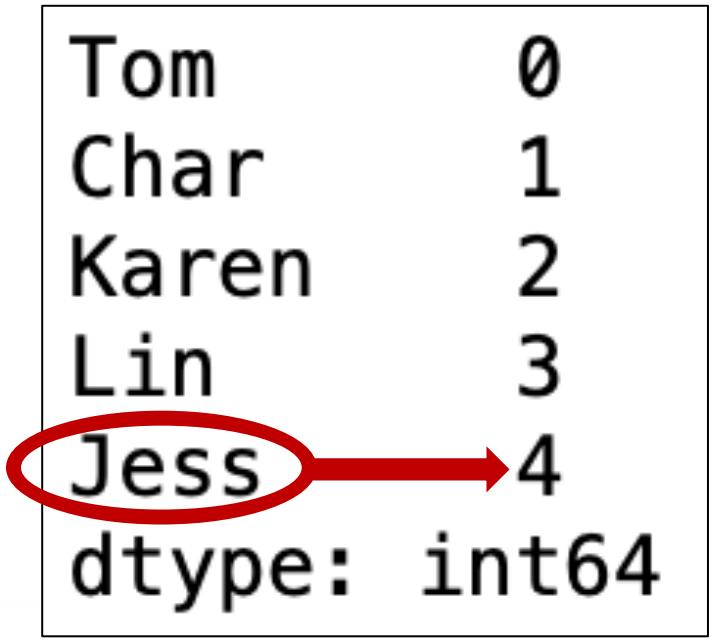
```
0    None
2    None
4    None
6    None
8    None
dtype: object
```

Indexes from a Numpy array

Indexing a Series (One Element)

- Get the 4th value (or the value corresponding to the index “Jess”) in the Pandas Series **s**)

s=



Equivalently:

```
s['Jess']
```

Or

```
s.loc['Jess']
```

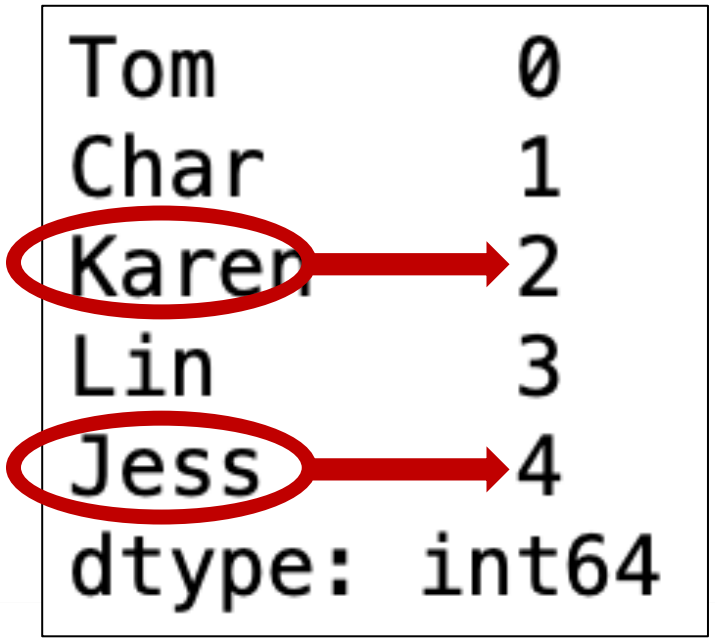
Or

```
s.iloc[4]
```

Indexing a Series (Multiple Elements)

- Get the sub-series containing the 2nd & 4th values (or the values corresponding to “Karen” and “Jess”) in the Pandas Series **s**)

s=



Equivalently:

```
s[['Char','Jess']]
```

Or

```
s.loc[['Char','Jess']]
```

Or

```
s.iloc[[1,4]]
```

Modifying a Series (One Element)

Before:

s=

```
0    NaN
2    NaN
4    NaN
6    NaN
8    NaN
dtype: float64
```

```
s[4] = 1
```

After:

s=

```
0    NaN
2    NaN
4    1.0
6    NaN
8    NaN
dtype: float64
```

Modifying a Series (Multiple Elements)

Before:

s=

```
Tom      NaN
Char     NaN
Karen    NaN
Lin      NaN
Jess     NaN
dtype: float64
```

```
s[['Tom','Karen']] = [0,3]
```

After:

s=

```
Tom      0.0
Char     NaN
Karen    3.0
Lin      NaN
Jess     NaN
dtype: float64
```

Unique Values of a Series

	`Series.unique()`
Argument	N/A
Returns	a NumPy array containing unique values in `Series`

s=

```
0     NaN
1   1.000
2   2.000
3     NaN
4   1.000
dtype: float32
```

```
s.unique()
```


```
array([nan,  1.,  2.], dtype=float32)
```

Frequencies Counts of a Series

	`Series.value_counts()`
Argument	N/A
Returns	a Pandas Series object

s=

```
0      NaN
1    1.000
2    2.000
3      NaN
4    1.000
dtype: float32
```

```
s.value_counts()
```

output

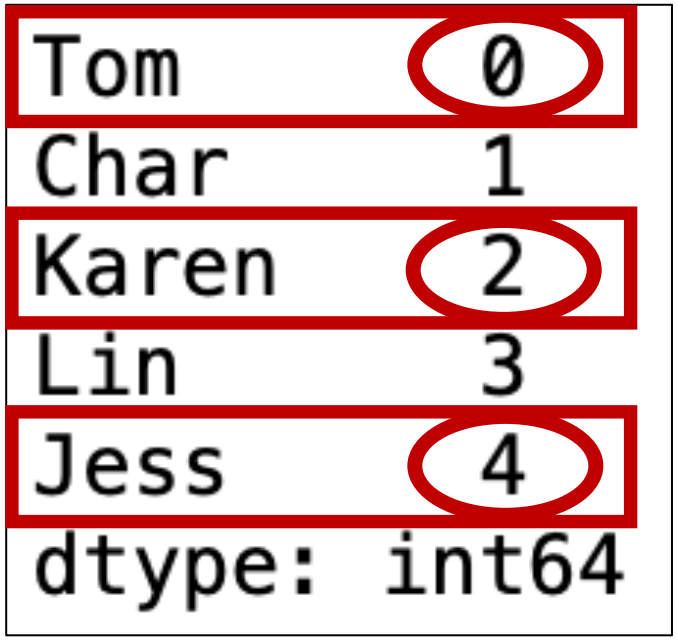
Index *Values*

```
1.000    2
2.000    1
Name: count, dtype: int64
```

Boolean Indexing & Filtering

- Get the sub-series with only values that are *even numbers* in **s** (Hint: recall Boolean indexing of an NumPy array, this is similar)

s=



Equivalently:

```
mask = (s%2==0)
s[mask]
```

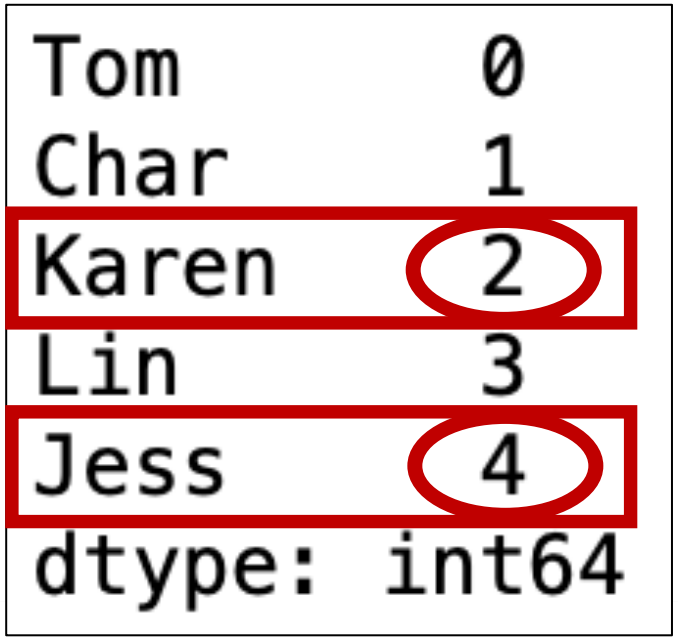
Or do it in one step

```
s[s%2==0]
```

Boolean Indexing & Filtering

- Get the sub-series with only values that are either 2 or 4 in **s** (Hint: use `Series.isin()`)

s=



Equivalently:

```
mask = s.isin([2,4])
s[mask]
```

Or do all in one step

```
s[s.isin([2,4])]
```

Convert a Series Into a Dictionary

	Series.to_dict()
Argument	N/A
Returns	a dictionary where the Series index becomes the dictionary keys, and the Series values become the dictionary values

s=

```
Tom      0
Char     1
Karen    2
Lin      3
Jess     4
dtype: int64
```

```
s.to_dict()
```

output

```
{'Tom': 0, 'Char': 1, 'Karen': 2, 'Lin': 3, 'Jess': 4}
```

Convert a Series Into a List

	Series.to_list()
Argument	N/A
Returns	a list containing the Series values as elements

s=

```
Tom      0
Char     1
Karen    2
Lin      3
Jess     4
dtype: int64
```

```
s.to_list()
```

output

```
[0, 1, 2, 3, 4]
```

Exercise: Working With a Pandas Series

1. Generate a Pandas Series named **coin** with 100 elements. Each element should be randomly drawn from a Bernoulli distribution with a mean of 0.8 (values are either 0 or 1). Set the index of the Series to integers from 1 to 100
2. Replace the elements at the 5th, 50th, 65th, and 90th positions with NaN
3. Identify and list all unique values in the Series, including NaN
4. Count the distinct values in the Series, including NaN
5. Create a new Series that retains only the 1 values and NaN from the original Series (excluding 0)
6. Convert the new Series into a dictionary

Broadcasting

```
s[[5,50,65,90]] = np.nan
```

- Is this syntax correct? Why?

```
L = [0,1,2,3,4]
L[2:3]=np.nan
```

- Is this syntax correct? Why?

Distinct Types of Missing Values

`np.nan`, `None`, `float('nan')`, and `math.nan` are distinct kinds of missing values, while `''` (empty string) is not a missing value

- NOT equivalent despite `pd.isna()` evaluating to `True` for all
- Note `pd.isna()` different from `np.isnan()` which only works on numeric types (e.g., `np.isnan(None)` returns an error)

Detect Missing Values in a Series

	`pd.isna(argument)`
Argument	An array-like object (Series, NumPy array, list, etc)
Returns	a Boolean Series of the same size as *argument*: `True` if the value in the original Series is missing (`NaN` [numerical] or `None` [string]), and `False` otherwise
	`Series.isna()`
Argument	N/A
Returns	a Boolean Series of the same size as *Series*: `True` if the value in the original Series is missing (`NaN` [numerical] or `None` [string]), and `False` otherwise

Detect Missing Values in a Series

	`pd.notna(argument)`
Argument	An array-like object (Series, NumPy array, list, etc)
Returns	a Boolean Series of the same size as *argument*: `True` if the value in the original Series is *NOT* missing (`NaN` [numerical] or `None` [string]), and `False` otherwise
	`Series.notna()`
Argument	a Pandas Series object
Returns	a Boolean Series of the same size as *Series*: `True` if the value in the original Series is *NOT* missing (`NaN` [numerical] or `None` [string]), and `False` otherwise

Drop Missing Values in a Series

	`Series.dropna()`
Argument	N/A
Returns	A new Series equal to the original Series with missing values dropped

	`Series.dropna(inplace=True)`
Argument	N/A
Returns	None [Series is directly modified, with missing values dropped without returning a new object]

Fill Missing Values in a Series

	`Series.fillna(argument)`
Argument	a value of the same data type as the original Series
Returns	A new Series equal to the original Series with missing values filled by the parameter passed as the argument

	`Series.fillna(argument,inplace=True)`
Argument	a value of the same type as the original Series
Returns	None [Series is directly modified, with missing values filled in place without returning a new object]

Functions vs. `inplace` Methods

Sometimes, a function and a method of the same name carry out the same operation (e.g., `isna()` and `notna()`)

- The function return the modified object as a new object, and the original object remains unchanged

```
df = pd.func(df)
```

- The method with `inplace=False` (default) same as the function

```
df = df.func(inplace=False)
```

- The method with `inplace=True` (passed as argument) directly modify the original object

```
df.func(inplace=True)
```

Inplace Method to Fill Missing Data

Before:

```
s.fillna(0,inplace=True)
```

After:

s=

```
Tom      1.0
Char     NaN
Karen    3.0
Lin      NaN
Jess     NaN
dtype: float64
```

s=

```
Tom      1.0
Char     0.0
Karen    3.0
Lin      0.0
Jess     0.0
dtype: float64
```

Inplace Method to Fill Missing Data

Before:

```
s.fillna('nn',inplace=True)
```

After:

s=

```
Tom          a
Char      None
Karen        c
Lin       None
Jess      None
dtype: object
```

s=

```
Tom          a
Char        nn
Karen        c
Lin         nn
Jess        nn
dtype: object
```

Exercise: Working With Missing Values

1. Generate a Pandas Series with 100 numbers sampled from a standard normal distribution *N(0,1)*
2. Set the first 10 numbers in the Series to `np.nan`. Set the next 10 numbers to `None`
3. Create a new Series that contains only the non-missing values from the original Series
4. Create a new Series where all missing values in the original Series are replaced with the sample mean
5. Replace all missing values in the original Series with 0 using an *inplace method*