Pandas DataFrame

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
```

Combine Row Series Into DataFrame

Concatenate row Series (i.e., realized values of all variables for the same observation) into a Pandas DataFrame

```
row_0 = pd.Series(['Char',10,False],
index=['name','age','isFemale’])

row_1 = pd.Series(['Lin',-1,True],
index=['name','age','isFemale’])


pd.DataFrame([row_0,row_1])
```

output

```
name          Char
age             10
isFemale     False
dtype: object
```

```
name          Lin
age            -1
isFemale     True
dtype: object
```

Stack the rows

Pandas DataFrame

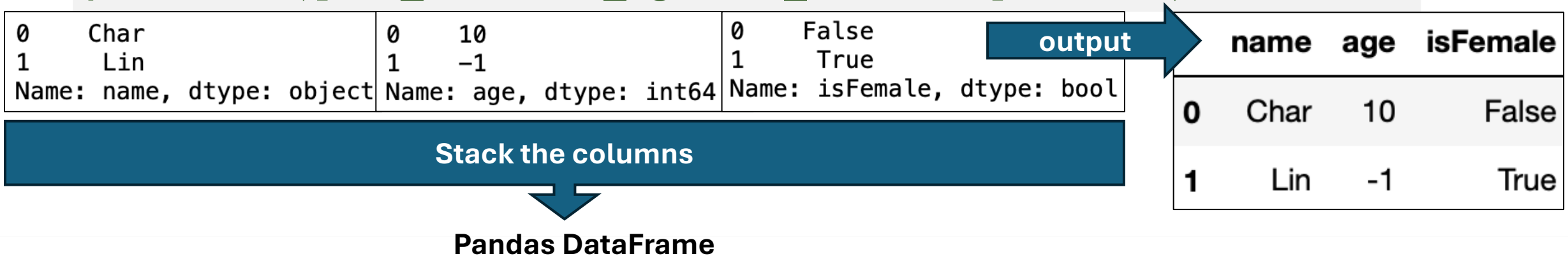
	name	age	isFemale
0	Char	10	False
1	Lin	-1	True

Combine Column Series Into DataFrame

Concatenate column Series (i.e., realized values of the same variable for all observations) into a Pandas DataFrame

```
col_name = pd.Series(['Char','Lin'],name='name’)
col_age = pd.Series([10,-1],name='age’)
col_isFemale = pd.Series([False,True],name='isFemale')

pd.concat([col_name,col_age,col_isFemale],axis=1)
```



Traverse a DataFrame Row by Row

```
for index, row in df.iterrows():
    ...
```

`row` is a Pandas Series, and **`index`** is the name of that Series

df=

	name	age	isFemale
0	Char	10	False
1	Lin	-1	True

index=0 row=

```
name          Char
age             10
isFemale     False
Name: 0, dtype: object
```

index=1 row=

```
name          Lin
age            -1
isFemale     True
Name: 1, dtype: object
```

Pandas DataFrame: Review

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

Column names *(`df.columns`)*

df=

	name	age	isFemale
0	Char	10	False
1	Lin	-1	True

Row index *(`df.index`)*

Q: Must column names and row indexes be unique?

Q: Is each column or row a Pandas Series?

Read & Create a DataFrame

Input Sources for Pandas DataFrame

- Files containing **tabular data**: e.g., CSV files, Stata files
- Files storing **semi-structure data**: e.g., JSON files, HTML files
- Built-in Python data types:
  - **Array-like objects:** e.g., lists, NumPy arrays
  - **Dictionaries**

Read Tabular Data Into a DataFrame

`pd.read_csv(argument)`	
Arguments	string containing the path and name of the CSV file
Returns	a Pandas DataFrame containing the stored data

`pd.read_csv(arg, chunksize=N)`	
Arguments	`arg`: string containing path and name of the CSV file `N`: number of rows to read in each iteration
Returns	(iterate through the file until reaching the end of file) a Pandas DataFrame containing `N` rows of the data

Note: The second approach of reading the data as a sequence of smaller chunks is very useful for processing very large CSV files (e.g., with more than 1 million rows)

Read Tabular Data Into a DataFrame

- **From a CSV (*.csv) file**

```
df = pd.read_csv('pathtofile/f.csv')
```

➢For very large CSV files, read the file in chunks

```
for chunk in pd.read_csv('pathtofile/f.csv',chunksize=N):
    ...
```

- **From a Stata (*.dta) file**

```
df = pd.read_stata('pathtofile/filename.dta')
```

➢For very large Stata files, read the file in chunks

```
for chunk in pd.read_stata('pathtofile/f.dta',chunksize=N):
    ...
```

Create a DataFrame from Built-In Types

`pd.DataFrame(arg,columns=c,index=d)`	
Arguments	`arg`: an array-like object (e.g., list) or a dictionary c: None (default) or a 1D array-like object (e.g., list) d: None (default) or a 1D array-like object (e.g., list)
Returns	a Pandas DataFrame containing the stored data

`pd.DataFrame.from_dict(arg,orient='index', _otheroptions)`	
Arguments	`arg`: a dictionary
Returns	a Pandas DataFrame containing the stored data

Create a DataFrame from Built-In Types

Most common ways to create a Pandas DataFrame:

- From a *dictionary of column arrays* (e.g., lists, series)
- From a *list of row dictionaries*
- From a *2D matrix (with column names)*
- From a *2D dictionary*
- From a *JSON string*

From *a dictionary of column arrays*

`data= {'col0':array_col0,...}` where `array_col0,...` is any array-like object such as list, Pandas Series, NumPy array

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

- Example:

```
>> dict_cols=
{'name':['Thomas','Charles','Karen','Linda','Jessica','William','Su
san','John','David','Jennifer'],
'ID':[10000,10001,10002,10003,10004,10005,10006,10007,10008,10009]}

>> df = pd.DataFrame(dict_cols)
```

From *a list of row dictionaries*

`data= [{'col0':v00,...,'coln':v0n},...]` where each element is a dictionary that maps variables col0,...coln to values in one observation

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

- Example:

```
>> list_rows =
[{'name':'Thomas','ID':10000},{'name':'Charles','ID':10001},{'name':'Kare
n','ID':10002},{'name':'Linda','ID':10003},{'name':'Jessica','ID':10004},
{'name':'William','ID':10005},{'name':'Susan','ID':10006},{'name':'John',
'ID':10007},{'name':'David','ID':10008},{'name':'Jennifer','ID':10009}]

>> df = pd.DataFrame(list_rows)
```

From *a 2D matrix (with column names)*

The rectangular matrix `data=[[v00,v01,...],...]` and column names `colnames=[col0,...]` with **N** observations and **K** variables

- If `data` is a list of *rows* (**N**x**K**)

```
df = pd.DataFrame(data,columns=colnames)
```

- If `data` is a list of *columns* (**K**x**N**)

```
df = pd.DataFrame(data).T
df.columns=colnames
```

From *a 2D matrix (with column names)*

The rectangular matrix `data=[[v00,v01,...],...]` and column names `colnames=[col0,...]` with **N** observations and **K** variables

- Example: data is a list of rows

```
>> list_rows =
[['Thomas',10000],['Charles',10001],['Karen',10002],['Linda',10003],['Jessica',10
004],['William',10005],['Susan',10006],['John',10007],['David',10008],['Jennifer'
,10009]]
>> df = pd.DataFrame(list_rows,columns=['name','ID'])
```

- Example: data is a list of columns

```
>> list_cols =
[['Thomas','Charles','Karen','Linda','Jessica','William','Susan','John','David','
Jennifer'],[10000,10001,10002,10003,10004,10005,10006,10007,10008,10009]]
>> df = pd.DataFrame(list_cols).T
>> df.columns=['name','ID']
```

From a *2D dictionary*

`data= {index0:{'col0':v00,...},...}` is a dictionary of row dictionaries where each first-level item represents an observation, with row index as the key and column dictionary as the value

```
df = pd.DataFrame.from_dict(data,orient='index’)
```

or equivalently,

```
df = pd.DataFrame(data).T
```

From a *2D dictionary*

`data= {index0:{'col0':v00,...},...}` is a dictionary of row dictionaries where each first-level item represents an observation, with row index as the key and column dictionary as the value

Example:

```
>> dict_rows =
{0:{'name':'Thomas','ID':10000},1:{'name':'Charles','ID':10001},2:{'name':'Karen'
,'ID':10002},3:{'name':'Linda','ID':10003},4:{'name':'Jessica','ID':10004},5:{'na
me':'William','ID':10005},6:{'name':'Susan','ID':10006},7:{'name':'John','ID':100
07},8:{'name':'David','ID':10008},9:{'name':'Jennifer','ID':10009}}

>> df = pd.DataFrame.from_dict(dict_rows,orient='index’)
>> df_equiv = pd.DataFrame(dict_rows).T
```

Create DataFrame from a JSON string

- An example JSON string (web data)

```
>> json_text =
'[{"@context":"http://schema.org","@type":"WebApplication","name":"Notion
World","description":"Discover the powerful world of Notion with this free directory
of the best resources and tools about Notion. Whether you are a beginner or want to
boost your Notion skills, this curated list will help you find everything you need
to make you a Notion master.","datePublished":"2022-09-26T13:34:12.815-
07:00","aggregateRating":{"@type":"AggregateRating","ratingCount":6,"ratingValue":"5
.0","worstRating":1,"bestRating":5},"offers":{"@type":"Offer","price":0,"priceCurren
cy":"USD"},"applicationCategory":"Productivity"},{"@context":"http://schema.org","@t
ype":"BreadcrumbList","itemListElement":[{"@type":"ListItem","position":1,"name":"Ho
me","item":"https://www.producthunt.com/"},{"@type":"ListItem","position":2,"name":"
Notion World","item":"https://www.producthunt.com/products/notion-world"}]}]'
```

Create DataFrame from a JSON string

Step 1. Parse the JSON string into a (nested) dictionary or list

	json.loads(text)
Argument	a `string` type object satisfying certain syntax requirement
Returns	a dictionary or a list containing the (parsed) JSON data

Requires the `json` module

```
>> import json
>> json.parsed = json.loads(json_text)

>> type(json_parsed)
```

output `list`

Note: `json_text` must be a **valid JSON string**, otherwise `json.loads(json_text)` cannot parse it and returns an error instead

Create DataFrame from a JSON string

Step 2. Examine the parsed JSON data

```
[{'@context': 'http://schema.org',
  '@type': 'WebApplication',
  'name': 'Notion World',
  'description': 'Discover the powerful world of Notion with this free directory of the best resources and tools ab
out Notion. Whether you are a beginner or want to boost your Notion skills, this curated list will help you find ev
erything you need to make you a Notion master.',
  'datePublished': '2022-09-26T13:34:12.815-07:00',
  'aggregateRating': {'@type': 'AggregateRating',
   'ratingCount': 6,
   'ratingValue': '5.0',
   'worstRating': 1,
   'bestRating': 5},
  'offers': {'@type': 'Offer', 'price': 0, 'priceCurrency': 'USD'},
  'applicationCategory': 'Productivity'},
 {'@context': 'http://schema.org',
  '@type': 'BreadcrumbList',
  'itemListElement': [{'@type': 'ListItem',
    'position': 1,
    'name': 'Home',
    'item': 'https://www.producthunt.com/'},
   {'@type': 'ListItem',
    'position': 2,
    'name': 'Notion World',
    'item': 'https://www.producthunt.com/products/notion-world'}]}]
```

→ The parsed JSON object is a list of nested dictionaries. Try to explore the data, for example:

```
>> len(json_parsed)
>> type(json_parsed[0])
>> json_parsed[0].keys()
>> json_parsed[0]['aggregateRating']
```

Create DataFrame from a JSON string

Step 3. Create a Pandas DataFrame from the parsed JSON object

```
pd.DataFrame(json_parsed)
```

output

	@context	@type	name	description	datePublished	aggregateRating	offers	applicationCategory	itemListElement
0	http://schema.org	WebApplication	Notion World	Discover the powerful world of Notion with thi...	2022-09-26T13:34:12.815-07:00	{'@type': 'AggregateRating', 'ratingCount': 6,...	{'@type': 'Offer', 'price': 0, 'priceCurrency'...	Productivity	NaN
1	http://schema.org	BreadcrumbList	NaN	NaN	NaN	NaN	NaN	NaN	[{'@type': 'ListItem', 'position': 1, 'name': ...

Summary: JSON string → nested dictionary or list → pd.DataFrame. But resulting data contain values of complex data types, and **further data wrangling** *(next time)* is needed, e.g., aggregateRating[0] is a dictionary itemListElement[1] is a list

All Roads Lead to Rome

- There are often more than one way to read the same input data
- Make sure resulting DataFrame has the right structure and data
- Here are common problems that can go wrong (and how to fix)
  - *Flipped columns & rows*: fix this by switching columns and rows, i.e., transposing the original DataFrame df using **`df.T`**
  - *No column names*: fix this by assigning column names to **`df.columns`**
  - *Row indexes are not 0 to N-1*: fix this by using **`df.reset_index()`** to reset indexes to default (0 to N-1)

Exercise: Read CSV and JSON Data

1. Read the file **census_data.csv** in chunks of 50 rows at a time using Pandas. Display each chunk as a separate DataFrame.

2. Copy the raw JSON string (`json_text`) from "Create a DataFrame from a JSON string" section and parse it into a JSON object
    1. Identify all **nested elements** in the JSON object (e.g., dictionaries or lists)
    2. Convert each nested element into a separate DataFrame (hint: *AggregateRating*, *Offer*, and *ListItem*)
    3. Get separate row Series from all the DataFrames, and combine them into a single DataFrame (hint: use `df.iterrows()`)
    4. Reset the combined DataFrame's indexes so that they are unique (hint: use `df.reset_index(drop=True)` – we'll learn more next session)