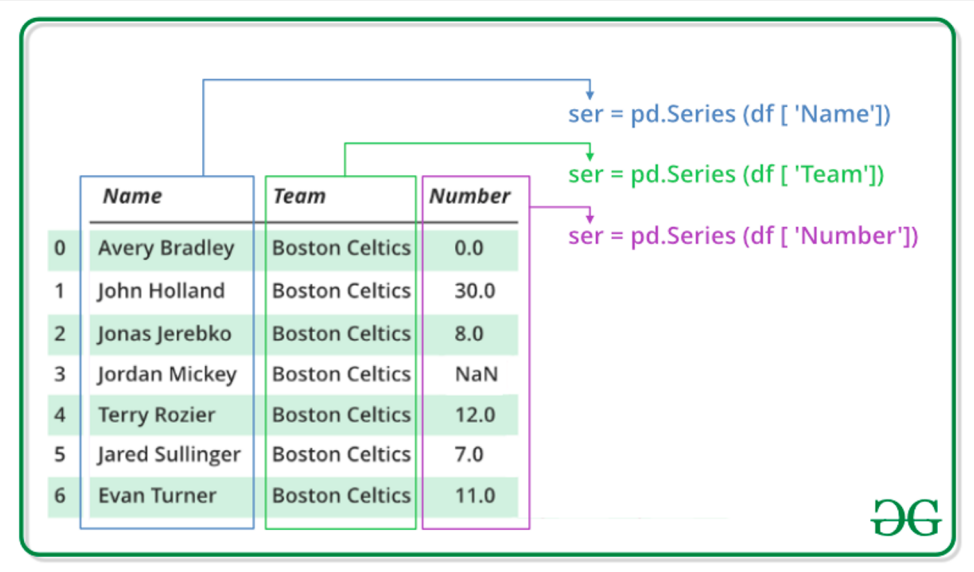
Intro to Pandas

Week 3

What is pandas?

Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.



Numpy vs Pandas

Numpy:

- Indexing
- Entire ndarray use the same data type

Pandas:

- Labeling + indexing
- Each column use the same data type

Basic Data Structures in Pandas

Series

- 1D
- Same data type
- Indexing only

DataFrame

- 2D
- Each column use the same data
- Indexing + Labeling

index	price
0	12.1
1	13.5
2	10.2
3	10.1
4	10.3
5	17.2

Series

index	price	color	size
0	12.1	red	M
1	13.5	black	L
2	10.2	black	XL
3	10.1	red	XL
4	10.3	green	M
5	17.2	yellow	L

DataFrame

Creating a DataFrame

read_csv( )

read_json( )

read_html( )

read_excel( )

read_feather( )

read_stata( )

read_sql( )

……

You can also:

Create using python list;

Create using python dictionaries (i.e: {“age”: [31,32,33]})

read_csv( ) function

In [9]:
```
# this file is seperated by tabs
# save to a variable
df = pandas.read_csv('../data/gapminder.tsv', sep='\t')
df.head()
```

Out[9]:

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106

info( ) function

A glance on the data information

```
>>> df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
country      1704 non-null object
continent    1704 non-null object
year         1704 non-null int64
lifeExp      1704 non-null float64
pop          1704 non-null int64
gdpPercap    1704 non-null float64
dtypes: float64(2), int64(2), object(2)
memory usage: 80.0+ KB
```

Data Selection

Dictionary style data selection (df['col_name'])

Accessing 1D data → pd.Series

```
>>> df['age']
0        21
1        31
2        33
3        52
4        12
        ...
20812    53
20813    24
20814    26
20815    18
20816    62
Name: age, Length: 20817, dtype:
int64
```

Accessing 2D data → pd.DataFrame

```
>>> df[['age']]
                    age
0                    21
1                    31
2                    33
3                    52
4                    12
...                 ...
20812    53
20813    24
20814    26
20815    18
20816    62

20817 rows × 1 columns
```

Numpy style data selection (df.iloc[row_idx, col_idx])

Getting a single item

```
>>> df.iloc[2, 5]
33
```

Accessing 1D data→ pd.Series

```
>>> df.iloc[2:5, 5]
2    33
3    52
4    12
Name: age, dtype: int64

>>> df.iloc[2, 2:4]
age               33
gender   M
Name: 2, dtype: object
```

Accessing 2D data→ pd.DataFrame

```
>>> df.iloc[2:5, 2:4]
        age      gender
2       33       M
3       52       F
4       12       F
```

label based indexing (df.loc[row_label, col_label])

Getting a single item

```
>>> df.loc[2, 'age']
33
```

Accessing 1D data→ pd.Series

```
>>> df.loc[2:4, 'age']
2    33
3    52
4    12
Name: age, dtype: int64

>>> df.loc[2, ['age', 'gender']]
age              33
gender   M
Name: 2, dtype: object
```

Accessing 2D data→ pd.DataFrame

```
>>> df.loc[2:4, ['age', 'gender']]
        age      gender
2       33       M
3       52       F
4       12       F
```

Data Selection Summary

Code	Return type	Description
`df['area']`	Pandas Series	One-dimensional data
`df[['area']]`	Pandas DataFrame	Two-dimensional, one column
`df.loc[2:, 'bedrooms' ]`	Pandas Series	One-dimensional data
`df.loc[2:, ['bedrooms'] ]`	Pandas DataFrame	Two-dimensional
`df.iloc[2, 4]`	Data's type stored at [2,4]	Only get 1 data
`df.iloc[2:5, 4]`	Pandas Series	One-dimensional data
`df.iloc[2:5, 4:8]`	Pandas DataFrame	Two-dimensional data

Data selection with boolean (Filtering)

Filter flights departure after 9:00

```
>>> df['dep_time'] > 900
0          True
1          True
2          True
3         False
4          True
          ...
20812     False
20813      True
20814     False
20815     False
20816      True
Name: dep_time, Length: 20817,
dtype: bool
```

Filter flights departure after 9:00, arrive before 15:00

```
>>> filter1 = df['dep_time'] > 900
>>> filter2 = df['arr_time'] < 1500
>>> df[(filter1) & (filter2)]
```

	fl_date	unique_carrier	airline_id	tail_num	fl_num	origin	dest	dep_time	dep_delay	arr_time	arr_delay	cancelled
0	2014-01-01 00:00:00	AA	19805	N338AA	1	JFK	LAX	914.00	14.00	1,238.00	13.00	0.00
5	2014-01-01 00:00:00	AA	19805	N323AA	185	JFK	LAX	2,133.00	-2.00	37.00	-18.00	0.00
10	2014-01-01 00:00:00	AA	19805	N3JWAA	178	JFK	BOS	1,253.00	3.00	1,351.00	1.00	0.00
21	2014-01-01 00:00:00	AA	19805	N3KDAA	317	LGA	ORD	1,029.00	44.00	1,212.00	42.00	0.00
22	2014-01-01 00:00:00	AA	19805	N482AA	325	LGA	ORD	1,018.00	3.00	1,202.00	12.00	0.00
...	...	...	...	...	...	...	...	...	...	...	...	...
20778	2014-01-31 00:00:00	UA	19977	N512UA	274	JFK	LAX	1,123.00	-6.00	1,438.00	-1.00	0.00
20792	2014-01-31 00:00:00	UA	19977	N474UA	472	LGA	IAH	955.00	-10.00	1,251.00	-31.00	0.00
20800	2014-01-31 00:00:00	UA	19977	N546UA	642	JFK	SFO	1,023.00	-2.00	1,424.00	24.00	0.00

Exercise time!

Read the data set with the command:

```
import pandas as pd
url = "http://vincentarelbundock.github.io/Rdatasets/csv/ISLR/Auto.csv"
cars = pd.read_csv(url, index_col=0)
cars.head()
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
1	18.00	8	307.00	130	3504	12.00	70	1	chevrolet chevelle malibu
2	15.00	8	350.00	165	3693	11.50	70	1	buick skylark 320
3	18.00	8	318.00	150	3436	11.00	70	1	plymouth satellite
4	16.00	8	304.00	150	3433	12.00	70	1	amc rebel sst
5	17.00	8	302.00	140	3449	10.50	70	1	ford torino

Exercise time!

Your tasks 😊:

1. Select engine related 3 columns (“cylinders”, “displacement”, “horsepower”) from the dataset.
2. Select every 4th row.
3. Select rows with mpg > 30.
4. Display the number of cars have at least 30 MPG and at least 5 cylinders.

	mpg	cylinders	displacement	horsepower	weight	acceleration	year	origin	name
1	18.00	8	307.00	130	3504	12.00	70	1	chevrolet chevelle malibu
2	15.00	8	350.00	165	3693	11.50	70	1	buick skylark 320
3	18.00	8	318.00	150	3436	11.00	70	1	plymouth satellite
4	16.00	8	304.00	150	3433	12.00	70	1	amc rebel sst
5	17.00	8	302.00	140	3449	10.50	70	1	ford torino

Adding, changing and deleting data

Update/insert a row

Use loc to update a row

```
# list size has to match row size
>>> df.loc[200] = [1,2,3,4,5,'hi']
```

Use iloc to update a row

```
# Has to match row size
>>> df.iloc[200] = [1,2,3,4,5,'hi']
```

Use loc to insert a row

```
# list size has to match row size
# row number can be anything
>>> df.loc[99999] = [1,2,3,4,5,'hi']
```

Use DataFrame.append( ) to insert a row?

```
>>> df.append([[1,2,3,4,5,'hi']])
```

In practice, you should try to pre-allocate space to do insertions, avoid inserting new rows.

Because pandas copies everything for each insertion.

Update/insert a column

For columns, you should first create a Series object to represent a column

```
# Create a new pd.Series with zeroes
>>> new_col = pd.Series(0, index=df.index)
```

Use the dictionary style to update/insert a row

```
>>> df['new'] = new_col
```

Use the numpy style to update/insert a row

```
>>> df.loc[:, 'new'] = new_col
```

Deleting columns/rows

Use the drop function to delete
Pass in row labels/col labels!

```
DataFrame.drop(labels, axis=0,
level=None, inplace=False,
errors='raise')
```

Delete rows

```
>>> df.drop([2,3,4,5], axis = 0,
inplace = True)
```

Delete cols

```
>>> df.drop(['origin', 'dest'], axis
= 1, inplace = True)
```

Concatenate, Join and Merge

pd.concat( ) (Array style merge)

Docstring description for
pd.concat(objs, axis = 0, join = “outer”, ...)

```
"""
Concatenate pandas objects along a particular axis with
optional set logic along the other axes.
"""
```

- Creates a copy
- Default axis = 0, add new rows
- If axis = 1, add new cols
- “Inner” and “Outer” joins. Default outer join.

pd.concat(join = outer) example

```
>>> result = pd.concat([df1, df4], axis = 1, join = 'outer')
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D	F
0	A0	B0	C0	D0	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3
6	NaN	NaN	NaN	NaN	B6	D6	F6
7	NaN	NaN	NaN	NaN	B7	D7	F7

pd.concat(join = inner) example

```
>>> result = pd.concat([df1, df4], axis = 1, join = 'inner')
```

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df4

	B	D	F
2	B2	D2	F2
3	B3	D3	F3
6	B6	D6	F6
7	B7	D7	F7

Result

	A	B	C	D	B	D	F
2	A2	B2	C2	D2	B2	D2	F2
3	A3	B3	C3	D3	B3	D3	F3

pd.merge( ) (SQL style merge)

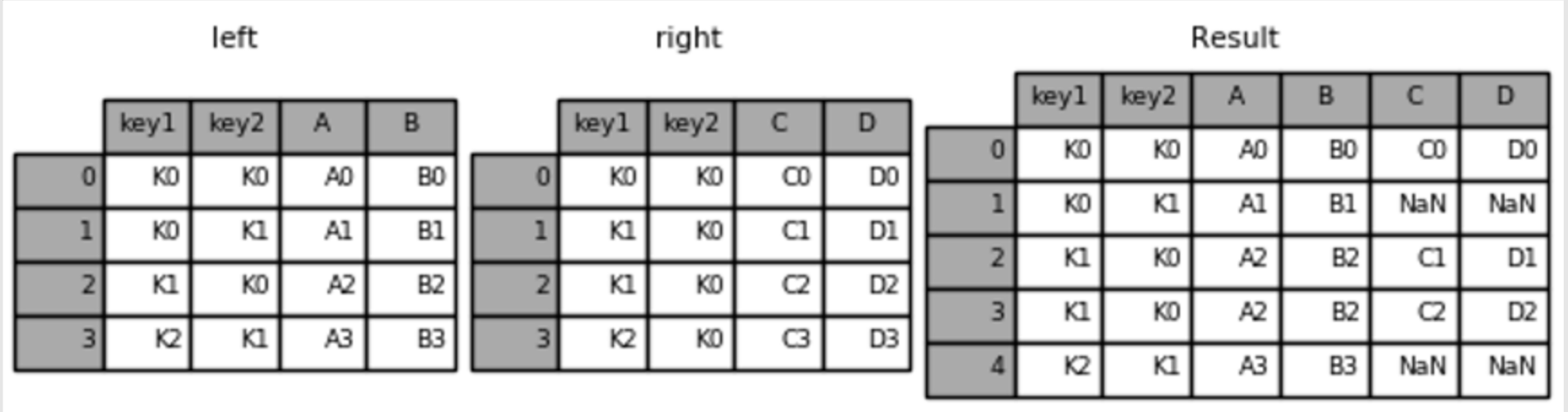
Docstring description for
pd.merge(left, right, how="inner", on=None, left_on=None, right_on=None ...)

```
"""
Merge DataFrame objects by performing a database-style
join operation by columns or indexes.
"""
```

how=	SQL Equivalent	Description
left	LEFT OUTER JOIN	Use keys from left only
right	RIGHT OUTER JOIN	Use keys from right only
inner	INNER JOIN	Intersection of left, right keys
outer	OUTER JOIN	Union of left, right keys

pd.merge(how = left) example

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



left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN

pd.merge(how = right) example

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

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	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

pd.merge(how = outer) example

```
>>> result = pd.merge(left, right, how='outer', keys=['key1','key2'])
```

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	key1	key2	A	B	C	D
0	K0	K0	A0	B0	C0	D0
1	K0	K1	A1	B1	NaN	NaN
2	K1	K0	A2	B2	C1	D1
3	K1	K0	A2	B2	C2	D2
4	K2	K1	A3	B3	NaN	NaN
5	K2	K0	NaN	NaN	C3	D3

pd.merge(how = inner) example

```
>>> result = pd.merge(left, right, how='inner', keys=['key1','key2'])
```

left

	key1	key2	A	B
0	K0	K0	A0	B0
1	K0	K1	A1	B1
2	K1	K0	A2	B2
3	K2	K1	A3	B3

right

	key1	key2	C	D
0	K0	K0	C0	D0
1	K1	K0	C1	D1
2	K1	K0	C2	D2
3	K2	K0	C3	D3

Result

	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.DataFrame.join( ) (SQL style merge)

Docstring description for
pd.DataFrame.join(other, on = None, how = “left”, ...)

```
"""
Join columns with other DataFrame either on index or on
a key column.
"""
```

- Similar to pd.merge( )
- By default, uses row labels to join.
  (Whereas pd.merge( ) uses cell values)

Exercise time!

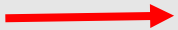
😄

```
url2 = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSBUNup2v-KIb528YJ6Qegwu69G9JpjpLQ3JzC8hicE-nIcHlF5MjNtxEdFdcAzefJ8h0MsTG_Sp1Yx/pub?gid=1224420987&single=true&output=csv"
url3 = "https://docs.google.com/spreadsheets/d/e/2PACX-1vSw7K-qOSFkIJJiil-ajQVYrVWL4poE4Wk3e73ZHVhJQaI-Qa_tfvS5suW0kPtjxfq6RxBNCNUxm9bT/pub?gid=1157486544&single=true&output=csv"
gdp = pd.read_csv(url2)
cpi = pd.read_csv(url3)
```

	DATE	GDP
0	1947-01-01	243.1
1	1947-04-01	246.3
2	1947-07-01	250.1
3	1947-10-01	260.3
4	1948-01-01	266.2

OUTER JOIN on DATE column

	DATE	CPIAUCSL
0	1947-01-01	21.48
1	1947-02-01	21.62
2	1947-03-01	22.00
3	1947-04-01	22.00
4	1947-05-01	21.95



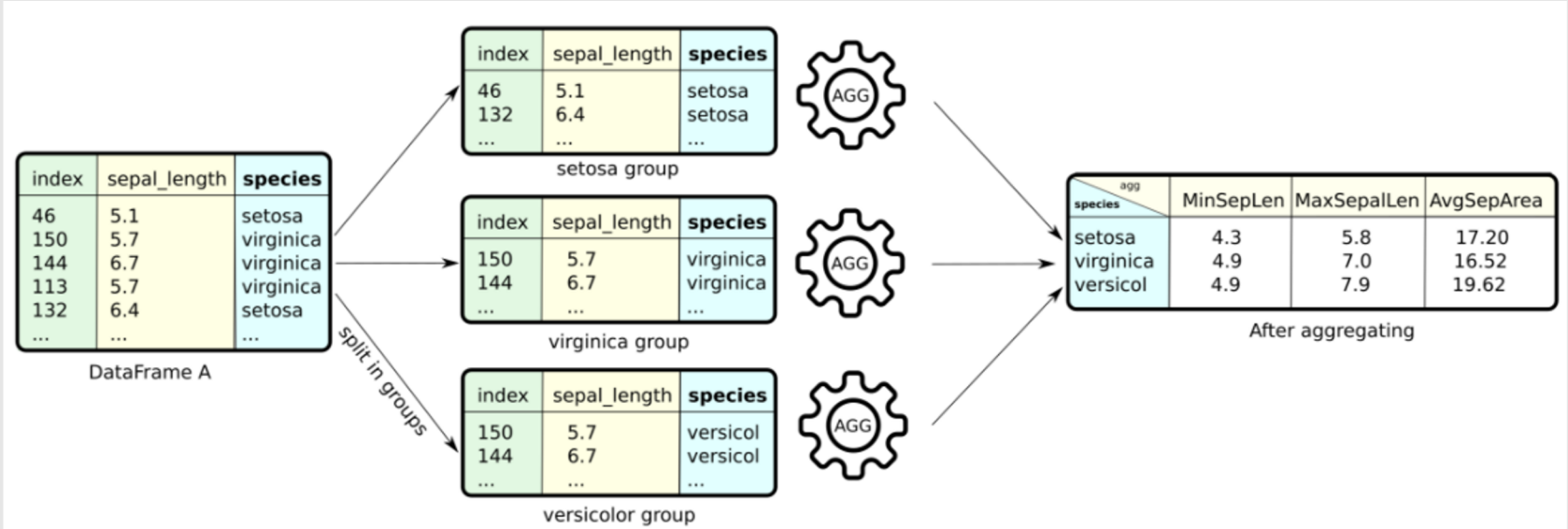
	DATE	GDP	CPIAUCSL
0	1947-01-01	243.1	21.48
1	1947-04-01	246.3	22.00
2	1947-07-01	250.1	22.23
3	1947-10-01	260.3	22.91
4	1948-01-01	266.2	23.68

Expected: (821, 3)

pd.DataFrame.groupby( )

Docstring description for
pd.DataFrame.groupby(by=None, axis = 0, ...)

```
"""
Group series using mapper (dict or key function, apply given function
to group, return result as series) or by a series of columns.
"""
```



Exercise time!

Use the following dataset:

```
url =
"https://raw.githubusercontent.com/justmarkham/DAT8/master/data/drinks.csv"
df = pd.read_csv(url, index_col=0)
df.head()
```

Question to solve: Which continent drinks more beer on average?

country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
Afghanistan	0	0	0	0.00	AS
Albania	89	132	54	4.90	EU
Algeria	25	0	14	0.70	AF
Andorra	245	138	312	12.40	EU
Angola	217	57	45	5.90	AF

pd.DataFrame.pivot_table( )

pd.DataFrame.pivot_table(values=None, index=None, columns=None, aggfunc='mean', ...)

Reshapes the DataFrame.

- index: what will the new index column be?
- columns: which columns do you want in your resulting table?
- values: which values will the table summarize?
- aggfunc: how should records be combined into a single statistic/number? (default mean)

Demo time! pivot the following table

```
url =
"https://raw.githubusercontent.com/codebasics/py/master/pandas/10_pivot/weather2.csv"
```

date	city	temperature	humidity
5/1/2017	new york	65	56
5/1/2017	new york	61	54
5/2/2017	new york	70	60
5/2/2017	new york	72	62
5/1/2017	mumbai	75	80
5/1/2017	mumbai	78	83
5/2/2017	mumbai	82	85
5/2/2017	mumbai	80	26

pd.DataFrame.describe( )

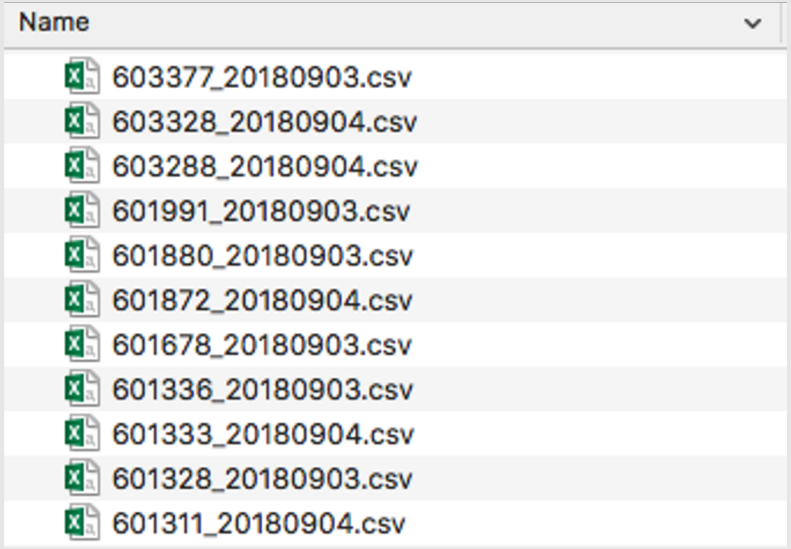
```
>>> df.describe( )
```

	airline_id	fl_num	dep_time	dep_delay	arr_time	arr_delay	cancelled
count	20,817.00	20,817.00	18,462.00	18,462.00	18,412.00	18,383.00	20,817.00
mean	20,109.61	1,826.10	1,319.99	22.77	1,493.70	21.38	0.12
std	370.72	1,548.19	480.00	59.77	518.85	64.61	0.32
min	19,393.00	1.00	1.00	-112.00	1.00	-112.00	0.00
25%	19,790.00	472.00	858.00	-4.00	1,110.75	-12.00	0.00
50%	20,355.00	1,457.00	1,336.00	0.00	1,519.00	3.00	0.00
75%	20,409.00	2,701.00	1,720.00	22.00	1,923.00	28.00	0.00
max	21,171.00	6,258.00	2,400.00	973.00	2,400.00	996.00	1.00

pd.DataFrame.to_csv( )

```
>>> df.to_csv('{}_20180903.csv'.format(stock_ID), index='False')

>>> df.to_csv('{}_20180904.csv'.format(stock_ID), index='False')
```



pd.DataFrame.fillna( )

```
>>> df.fillna(value = None, method = None)
```

Fill methods:

- **ffill** (Use i - 1 row's value)
- **bfill** (Use i + 1 row's value)
- **None** (Use the specified fill value)

pd.DataFrame/Series.astype( )

```
>>> df.astype(dtype, copy=True, errors='raise')
```

Cast entire DataFrame/Series to parameter dtype.