Foundations of Data Science

Master in Data Science

2022 / 2023

TABLE OPERATIONS

Tables

Special Column, called "Index", or "ID", or "Key"
Usually, no duplicates Allowed

Variables
(also called Attributes, or Columns, or Labels)

Observations, Rows, or Tuples

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Tables

ID	Address
1	College Park, MD, 20742
2	Washington, DC, 20001
3	Silver Spring, MD 20901

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085

1. Select/slicing

- Select only some of the rows, or some of the columns, or a combination

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Only columns ID and Age

ID	age
1	12.2
2	11.0
3	15.6
4	35.1

Only rows with wgt > 41

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Both

ID	age
1	12.2
3	15.6
4	35.1

2. Aggregate/Reduce

- Combine values across a column into a single value

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

SUM

73.9	**232.6**	**640.0**

MAX

35.1	**84.2**	**185.8**

SUM(wgt_kg^2 - hgt_cm)

14167.66

What about ID/Index column?

Usually not meaningful to aggregate across it

May need to explicitly add an ID column

3. Map

- Apply a function to every row, possibly creating more or fewer columns

ID	Address
1	College Park, MD, 20742
2	Washington, DC, 20001
3	Silver Spring, MD 20901

ID	City	State	Zipcode
1	College Park	MD	20742
2	Washington	DC	20001
3	Silver Spring	MD	20901

Variations that allow one row to generate multiple rows in the output (sometimes called “flatmap”)

4. Group By

- Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'A' →

A = foo

ID	B	C
1	3	6.6
3	4	3.1
4	3	8.0
7	4	2.3
8	3	8.0

A = bar

ID	B	C
2	2	4.7
5	1	1.2
6	2	2.5

4. Group By

- Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'B' →

B = 1

ID	A	C
5	bar	1.2

B = 2

ID	A	C
2	bar	4.7
6	bar	2.5

B = 3

ID	A	C
1	foo	6.6
4	foo	8.0
8	foo	8.0

B = 4

ID	A	C
3	foo	3.1
7	foo	2.3

4. Group By

- Group tuples together by column/dimension

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By ‘A’, ‘B’ →

A = bar, B = 1

ID	C
5	1.2

A = bar, B = 2

ID	C
2	4.7
6	2.5

A = foo, B = 3

ID	C
1	6.6
4	8.0
8	8.0

A = foo, B = 4

ID	C
3	3.1
7	2.3

5. Group By Aggregate

- Compute one aggregate per group

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group by ‘B’
Sum on C

B = 1

ID	A	C
5	bar	1.2

B = 1

Sum (C)
1.2

B = 2

ID	A	C
2	bar	4.7
6	bar	2.5

B = 2

Sum (C)
7.2

B = 3

ID	A	C
1	foo	6.6
4	foo	8.0
8	foo	8.0

B = 3

Sum (C)
22.6

B = 4

ID	A	C
3	foo	3.1
7	foo	2.3

B = 4

Sum (C)
5.4

5. Group By Aggregate

- Final result usually seen as a table

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group by ‘B’
Sum on C

B = 1

Sum (C)
1.2

B = 2

Sum (C)
7.2

B = 3

Sum (C)
22.6

B = 4

Sum (C)
5.4

B	SUM(C)
1	1.2
2	7.2
3	22.6
4	5.4

6. Union/Intersection/Difference

- Set operations – only if the two tables have identical attributes/columns

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0

U

ID	A	B	C
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

→

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

7. Merge or Join

- Combine rows/tuples across two tables if they have the same key

ID	A	B
1	foo	3
2	bar	2
3	foo	4
4	foo	3

$\bowtie$

ID	C
1	1.2
2	2.5
3	2.3
5	8.0

$\rightarrow$

ID	A	B	C
1	foo	3	1.2
2	bar	2	2.5
3	foo	4	2.3

What about IDs not present in both tables?

Often need to keep them around

Can "pad" with NaN

7. Merge or Join

- Combine rows/tuples across two tables if they have the same key
- Outer joins can be used to ”pad” IDs that don’t appear in both tables
- Three variants: LEFT, RIGHT, FULL
- SQL Terminology -- Pandas has these operations as well

ID	A	B
1	foo	3
2	bar	2
3	foo	4
4	foo	3

$⟗$

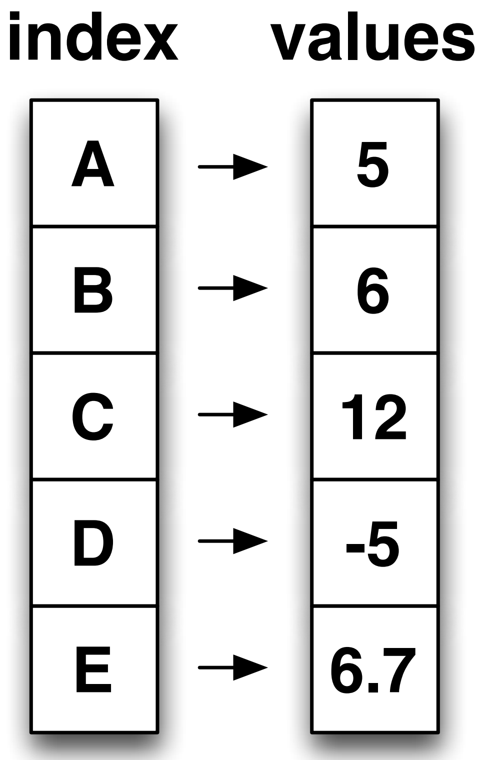
ID	C
1	1.2
2	2.5
3	2.3
5	8.0

ID	A	B	C
1	foo	3	1.2
2	bar	2	2.5
3	foo	4	2.3
4	foo	3	NaN
5	NaN	NaN	8.0

PANDAS

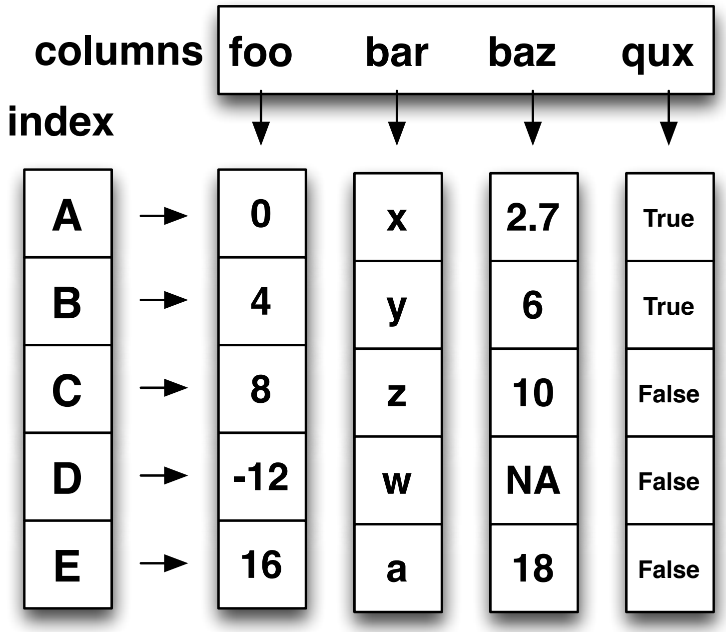
Pandas: Series

- Subclass of numpy.ndarray
- Data: any type
- Index labels need not be ordered
- Duplicates possible but result in reduced functionality



Pandas: DataFrame

- Each column can have a different type
- Row and Column index
- Mutable size: insert and delete columns



Index

```python
s = pd.Series(np.random.random(4))
s
```

✓ 0.3s

```
0    0.806355
1    0.285593
2    0.154172
3    0.480174
dtype: float64
```

```python
s.index
```

✓ 0.4s

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

```python
dummy_data = np.hstack([np.arange(21,27).reshape(6,1),
    np.random.random([6, 3])])
df = pd.DataFrame(dummy_data)
df
```

✓ 0.8s

	0	1	2	3
0	21.0	0.613801	0.176179	0.460320
1	22.0	0.658697	0.257065	0.868803
2	23.0	0.495096	0.629086	0.484089
3	24.0	0.360152	0.564157	0.911771
4	25.0	0.612299	0.387282	0.058688
5	26.0	0.629878	0.224060	0.899951

```python
df.index
```

✓ 0.9s

```
RangeIndex(start=0, stop=6, step=1)
```

Index

```
weekdays = ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday']
s2 = pd.Series(np.random.random(7), index=weekdays)
s2
```

✓ 0.6s

```
monday       0.408381
tuesday      0.062953
wednesday    0.931653
thursday     0.222275
friday       0.187165
saturday     0.701107
sunday       0.328268
dtype: float64
```

```
s2.index
```

✓ 0.4s

```
Index(['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday',
       'sunday'],
      dtype='object')
```

Index

```
df = pd.DataFrame(dummy_data[:, 1:], index = dummy_data[:, 0], columns=['A', 'B', 'C'])
df
```

✓ 0.6s

	A	B	C
21.0	0.613801	0.176179	0.460320
22.0	0.658697	0.257065	0.868803
23.0	0.495096	0.629086	0.484089
24.0	0.360152	0.564157	0.911771
25.0	0.612299	0.387282	0.058688
26.0	0.629878	0.224060	0.899951

Reindex

```
df.reindex(np.arange(23,30))
```

✓ 0.4s

	A	B	C
23	0.480339	0.074822	0.962559
24	0.134096	0.802457	0.022252
25	0.396679	0.869059	0.947367
26	0.241331	0.661447	0.016198
27	NaN	NaN	NaN
28	NaN	NaN	NaN
29	NaN	NaN	NaN

```
df.reindex(np.arange(23,30), method='ffill')
```

✓ 0.5s

	A	B	C
23	0.480339	0.074822	0.962559
24	0.134096	0.802457	0.022252
25	0.396679	0.869059	0.947367
26	0.241331	0.661447	0.016198
27	0.241331	0.661447	0.016198
28	0.241331	0.661447	0.016198
29	0.241331	0.661447	0.016198

Indexing and Slicing

Indexing on columns

```
df[['B','C']]
```

✓ 0.5s

	B	C
21.0	0.029455	0.785111
22.0	0.043754	0.661880
23.0	0.074822	0.962559
24.0	0.802457	0.022252
25.0	0.869059	0.947367
26.0	0.661447	0.016198

Boolean indexing

```
df[df['C']>0.5]
```

✓ 0.7s

	A	B	C
21.0	0.511241	0.029455	0.785111
22.0	0.370496	0.043754	0.661880
23.0	0.480339	0.074822	0.962559
25.0	0.396679	0.869059	0.947367

Indexing and Slicing

Selection: loc / iloc

```
df2 = pd.DataFrame(s2)
df2
```

✓ 0.8s

	Rain probability
monday	0.057201
tuesday	0.938698
wednesday	0.986811
thursday	0.663835
friday	0.081591
saturday	0.496326
sunday	0.273386

df.loc selects by label

```
df2.loc['friday']
```

✓ 0.3s

```
Rain probability    0.081591
Name: friday, dtype: float64
```

We can use a list or range

```
df2.loc['friday':'sunday']

# note the use of a range of labels, which results in the same as
# df2.loc[['friday', 'saturday', 'sunday']]
```

✓ 0.5s

	Rain probability
friday	0.081591
saturday	0.496326
sunday	0.273386

Indexing and Slicing

Selection: loc / iloc

```
df2 = pd.DataFrame(s2)
df2
```

✓ 0.8s

	Rain probability
monday	0.057201
tuesday	0.938698
wednesday	0.986811
thursday	0.663835
friday	0.081591
saturday	0.496326
sunday	0.273386

df.iloc selects by integer position

```
df2.iloc[4:]
```

✓ 0.6s

	Rain probability
friday	0.081591
saturday	0.496326
sunday	0.273386

```
df2.iloc[:3, 0]
```

✓ 0.6s

```
monday       0.057201
tuesday      0.938698
wednesday    0.986811
Name: Rain probability, dtype: float64
```

Indexing and Slicing

Selection: loc / iloc

```
df2 = pd.DataFrame(s2)
df2
```

✓ 0.8s

	Rain probability
monday	0.057201
tuesday	0.938698
wednesday	0.986811
thursday	0.663835
friday	0.081591
saturday	0.496326
sunday	0.273386

We can loc/iloc on rows, columns, or both

```
df.loc[:, ['A', 'C']]
```

	A	C
21.0	0.511241	0.785111
22.0	0.370496	0.661880
23.0	0.480339	0.962559
24.0	0.134096	0.022252
25.0	0.396679	0.947367
26.0	0.241331	0.016198

```
df.loc[25:, ['A', 'C']]
```

	A	C
25.0	0.396679	0.947367
26.0	0.241331	0.016198

Arithmetic operations

Arithmetic operations on Series and DataFrames are index aligned

```
df1 = pd.DataFrame({'A': np.arange(5), 'B': np.arange(1, 6),
    'C': np.arange(2, 7)}, index=list('abcde'))
df1
```

✓ 0.4s

	A	B	C
a	0	1	2
b	1	2	3
c	2	3	4
d	3	4	5
e	4	5	6

```
df2 = pd.DataFrame({'B': np.arange(10,14), 'C': np.arange(11, 15),
    'E': np.arange(12, 16)}, index=list('bcdf'))
df2
```

✓ 0.4s

	B	C	E
b	10	11	12
c	11	12	13
d	12	13	14
f	13	14	15

```
df1 + df2
```

✓ 0.5s

	A	B	C	E
a	NaN	NaN	NaN	NaN
b	NaN	12.0	14.0	NaN
c	NaN	14.0	16.0	NaN
d	NaN	16.0	18.0	NaN
e	NaN	NaN	NaN	NaN
f	NaN	NaN	NaN	NaN

We can use arithmetic method, which allow fill values

```
df1.add(df2, fill_value=0.0)
```

✓ 0.6s

	A	B	C	E
a	0.0	1.0	2.0	NaN
b	1.0	12.0	14.0	12.0
c	2.0	14.0	16.0	13.0
d	3.0	16.0	18.0	14.0
e	4.0	5.0	6.0	NaN
f	NaN	13.0	14.0	15.0

Data Cleaning

Missing values

```
>>> df = pd.DataFrame(dict(age=[5, 6, np.NaN],
...                        born=[pd.NaT, pd.Timestamp('1939-05-27'),
...                              pd.Timestamp('1940-04-25')],
...                        name=['Alfred', 'Batman', ''],
...                        toy=[None, 'Batmobile', 'Joker']))
>>> df
   age       born    name        toy
0  5.0        NaT  Alfred       None
1  6.0 1939-05-27  Batman  Batmobile
2  NaN 1940-04-25              Joker
```

```
>>> df.isna()
     age   born   name    toy
0  False   True  False   True
1  False  False  False  False
2   True  False  False  False
```

```
>>> df.notna()
     age   born  name    toy
0   True  False  True  False
1   True   True  True   True
2  False   True  True   True
```

Missing values can be informative !

(in some cases)

Remove missing values

```
>>> df = pd.DataFrame(dict(age=[5, 6, np.NaN],
...                        born=[pd.NaT, pd.Timestamp('1939-05-27'),
...                              pd.Timestamp('1940-04-25')],
...                        name=['Alfred', 'Batman', ''],
...                        toy=[None, 'Batmobile', 'Joker']))
>>> df
   age        born    name        toy
0  5.0         NaT  Alfred       None
1  6.0  1939-05-27  Batman  Batmobile
2  NaN  1940-04-25                Joker
```

Drop the rows where at least one element is missing.

```
>>> df.dropna()
     name        toy       born
1  Batman  Batmobile 1940-04-25
```

Drop the columns where at least one element is missing.

```
>>> df.dropna(axis='columns')
       name
0    Alfred
1    Batman
2  Catwoman
```

Drop the rows where all elements are missing.

```
>>> df.dropna(how='all')
       name        toy       born
0    Alfred        NaN        NaT
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
```

Remove missing values

```
>>> df = pd.DataFrame(dict(age=[5, 6, np.NaN],
...                        born=[pd.NaT, pd.Timestamp('1939-05-27'),
...                              pd.Timestamp('1940-04-25')],
...                        name=['Alfred', 'Batman', ''],
...                        toy=[None, 'Batmobile', 'Joker']))
>>> df
   age        born    name        toy
0  5.0         NaT  Alfred       None
1  6.0  1939-05-27  Batman  Batmobile
2  NaN  1940-04-25              Joker
```

Keep only the rows with at least 2 non-NA values.

```
>>> df.dropna(thresh=2)
       name        toy       born
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
```

Define in which columns to look for missing values.

```
>>> df.dropna(subset=['name', 'toy'])
       name        toy       born
1    Batman  Batmobile 1940-04-25
2  Catwoman   Bullwhip        NaT
```

Fill missing data

```
>>> df = pd.DataFrame([[np.nan, 2, np.nan, 0],
...                    [3, 4, np.nan, 1],
...                    [np.nan, np.nan, np.nan, 5],
...                    [np.nan, 3, np.nan, 4]],
...                   columns=list("ABCD"))
>>> df
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
3  NaN  3.0 NaN  4
```

Replace all NaN elements with 0s.

```
>>> df.fillna(0)
    A   B   C   D
0   0.0 2.0 0.0 0
1   3.0 4.0 0.0 1
2   0.0 0.0 0.0 5
3   0.0 3.0 0.0 4
```

We can also propagate non-null values forward or backward.

```
>>> df.fillna(method="ffill")
    A   B   C   D
0   NaN 2.0 NaN 0
1   3.0 4.0 NaN 1
2   3.0 4.0 NaN 5
3   3.0 3.0 NaN 4
```

Replace all NaN elements in column 'A', 'B', 'C', and 'D', with 0, 1, 2, and 3 respectively.

```
>>> values = {"A": 0, "B": 1, "C": 2, "D": 3}
>>> df.fillna(value=values)
    A   B   C   D
0   0.0 2.0 2.0 0
1   3.0 4.0 2.0 1
2   0.0 1.0 2.0 5
3   0.0 3.0 2.0 4
```

Check also:

```
df.fillna(method='ffill', limit=2)
df.fillna(df.mean())
```

Remove duplicates

```
>>> df = pd.DataFrame({
...     'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],
...     'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
...     'rating': [4, 4, 3.5, 15, 5]
... })
>>> df
    brand style  rating
0  Yum Yum   cup     4.0
1  Yum Yum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

By default, it removes duplicate rows based on all columns.

```
>>> df.drop_duplicates()
    brand style  rating
0  Yum Yum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

To remove duplicates on specific column(s), use `subset`.

```
>>> df.drop_duplicates(subset=['brand'])
    brand style  rating
0  Yum Yum   cup     4.0
2  Indomie   cup     3.5
```

To remove duplicates and keep last occurrences, use `keep`.

```
>>> df.drop_duplicates(subset=['brand', 'style'], keep='last')
    brand style  rating
1  Yum Yum   cup     4.0
2  Indomie   cup     3.5
4  Indomie  pack     5.0
```

Replace duplicates

```
>>> df = pd.DataFrame({
...     'brand': ['Yum Yum', 'Yum Yum', 'Indomie', 'Indomie', 'Indomie'],
...     'style': ['cup', 'cup', 'cup', 'pack', 'pack'],
...     'rating': [4, 4, 3.5, 15, 5]
... })
>>> df
    brand style  rating
0  Yum Yum   cup     4.0
1  Yum Yum   cup     4.0
2  Indomie   cup     3.5
3  Indomie  pack    15.0
4  Indomie  pack     5.0
```

```
df.groupby(['brand', 'style']).mean()
```

✓ 0.5s

		rating
brand	**style**	
Indomie	cup	3.5
	pack	10.0
Yum Yum	cup	4.0

```
df.groupby(['brand', 'style']).mean().reset_index()
```

✓ 0.5s

	brand	style	rating
0	Indomie	cup	3.5
1	Indomie	pack	10.0
2	Yum Yum	cup	4.0

Strings and regular expressions

```
data = pd.Series({'Dave': 'dave@google.com', 'Steve': 'steve@gmail.com',
                  'Rob': 'rob@gmail.com', 'Wes': np.nan})

pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'

data.str.findall(pattern, flags=re.IGNORECASE)
```

✓ 0.5s

```
Dave     [(dave, google, com)]
Steve    [(steve, gmail, com)]
Rob        [(rob, gmail, com)]
Wes                        NaN
dtype: object
```

```
data.str.extract(pattern, flags=re.IGNORECASE)
```

✓ 0.8s

	0	1	2
Dave	dave	google	com
Steve	steve	gmail	com
Rob	rob	gmail	com
Wes	NaN	NaN	NaN

DATA WRANGLING

Merge

```
>>> df1 = pd.DataFrame({'a': ['foo', 'bar'], 'b': [1, 2]})
>>> df2 = pd.DataFrame({'a': ['foo', 'baz'], 'c': [3, 4]})
>>> df1
      a  b
0   foo  1
1   bar  2
>>> df2
      a  c
0   foo  3
1   baz  4
```

```
>>> df1.merge(df2, how='inner', on='a')
      a  b  c
0   foo  1  3
```

```
>>> df1.merge(df2, how='left', on='a')
      a  b  c
0   foo  1  3.0
1   bar  2  NaN
```

Merge

```
>>> df1 = pd.DataFrame({'left': ['foo', 'bar']})
>>> df2 = pd.DataFrame({'right': [7, 8]})
>>> df1
    left
0   foo
1   bar
>>> df2
    right
0   7
1   8
```

```
>>> df1.merge(df2, how='cross')
   left  right
0   foo      7
1   foo      8
2   bar      7
3   bar      8
```

Merge

```
>>> df1
    lkey value
0   foo      1
1   bar      2
2   baz      3
3   foo      5
>>> df2
    rkey value
0   foo      5
1   bar      6
2   baz      7
3   foo      8
```

Merge df1 and df2 on the lkey and rkey columns. The value columns have the default suffixes, _x and _y, appended.

```
>>> df1.merge(df2, left_on='lkey', right_on='rkey')
  lkey  value_x rkey  value_y
0  foo        1  foo        5
1  foo        1  foo        8
2  foo        5  foo        5
3  foo        5  foo        8
4  bar        2  bar        6
5  baz        3  baz        7
```

Merge DataFrames df1 and df2 with specified left and right suffixes appended to any overlapping columns.

```
>>> df1.merge(df2, left_on='lkey', right_on='rkey',
...           suffixes=('_left', '_right'))
  lkey  value_left rkey  value_right
0  foo           1  foo            5
1  foo           1  foo            8
2  foo           5  foo            5
3  foo           5  foo            8
4  bar           2  bar            6
5  baz           3  baz            7
```

Concat

```python
df1 = pd.DataFrame([['a', 1], ['b', 2]], columns=['letter', 'number'])
df1
```

✓ 0.8s

	letter	number
0	a	1
1	b	2

```python
df3 = pd.DataFrame([['c', 3, 'cat'], ['d', 4, 'dog']], columns=['letter', 'number', 'animal'])
df3
```

✓ 0.5s

	letter	number	animal
0	c	3	cat
1	d	4	dog

```python
pd.concat([df1, df3], sort=False)
```

✓ 0.6s

	letter	number	animal
0	a	1	NaN
1	b	2	NaN
0	c	3	cat
1	d	4	dog

Concat

Combine `DataFrame` objects with overlapping columns and return only those that are shared by passing `inner` to the `join` keyword argument.

```
>>> pd.concat([df1, df3], join="inner")
  letter  number
0      a       1
1      b       2
0      c       3
1      d       4
```

Combine `DataFrame` objects horizontally along the x axis by passing in `axis=1`.

```
>>> df4 = pd.DataFrame([['bird', 'polly'], ['monkey', 'george']],
...                    columns=['animal', 'name'])
>>> pd.concat([df1, df4], axis=1)
  letter  number  animal    name
0      a       1    bird   polly
1      b       2  monkey  george
```