

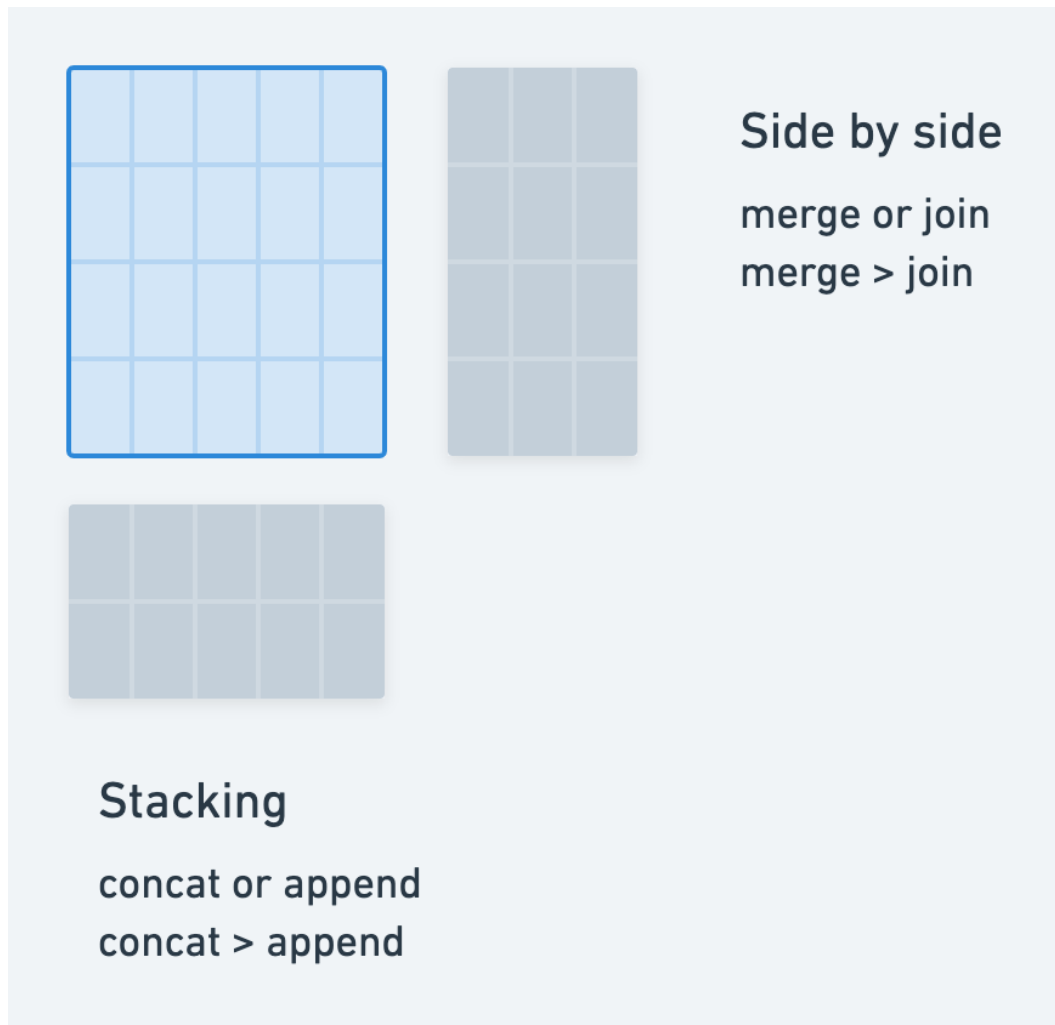


Data Wrangling

Agenda

- How to combine data in pandas
- Pandas datatypes and changing data types in data frames
- How to sort your data
- Pandas limitations and when not to use it
- Time to try out things out yourself

1. Combining Data



1.1 Concat

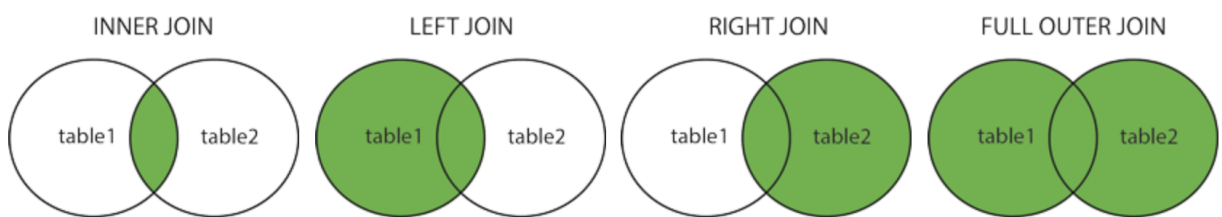
Mostly used to combine together observations with the same information (same columns) that are stored separately. While it can technically be used to add columns, however, we use merge for that.

Stock Price

<u>Aa</u> time	<u>≡</u> name	<u>#</u> close_price	<u>#</u> open_price
<u>12/01/1987</u>	microsoft	120	124
<u>13/01/1987</u>	google	131	132
<u>....</u>	...		

1.2 Merge

Used to join together data side by side whether for the same observations or different ones.



MERGE Inner/Outer

df1

Spice	In_stock	Ordered
Cinnamon	0	5
Anise	2	2
Nutmeg	3	0
Clove	7	0

`pd.merge(df1, df2, how= 'inner', on='Spice')`

Spice	In_stock	Ordered	Price
Cinnamon	0	5	3.50
Anise	2	2	4.00

df2

Spice	Price
Cinnamon	3.50
Anis	4.00
Saffron	9.90
Pepper	3.20
Turmeric	6.75

`pd.merge(df1, df2, how= 'outer', on='Spice')`

Spice	In_stock	Ordered	Price
Cinnamon	0	5	3.50
Anis	2	2	4.00
Nutmeg	3	0	NaN
Clove	7	0	NaN
Saffron	NaN	NaN	9.90
Pepper	NaN	NaN	3.20
Turmeric	NaN	NaN	6.75

MERGE left/right

df1

Spice	In_stock	Ordered
Cinnamon	0	5
Anise	2	2
Nutmeg	3	0
Clove	7	0

`pd.merge(left=df1, right=df2, how= 'left', on='Spice')`

Spice	In_stock	Ordered	Price
Cinnamon	0	5	3.50
Anise	2	2	4.00
Nutmeg	3	0	NaN
Clove	7	0	NaN

df2

Spice	Price
Cinnamon	3.50
Anis	4.00
Saffron	9.90
Pepper	3.20
Turmeric	6.75

`pd.merge(df1, df2, how= 'right', on='Spice')`

Spice	Price	In_stock	Ordered
Cinnamon	3.50	0	5
Anis	4.00	2	2
Saffron	9.90	NaN	NaN
Pepper	3.20	NaN	NaN
Turmeric	6.75	NaN	NaN



Merges dataframes on specific column/index with options left, right, inner, outer.

Tips:

- Be explicit about the `how=` and the `on=` attributes and avoid letting merge decide this for you

2. Data types

Pandas Datatypes

 Dtype	 Tags
<u>object</u>	text
<u>int64 or int32</u>	integer numbers
<u>float64 or float32</u>	floating point numbers
<u>bool</u>	True/False values
<u>category</u>	Finite list of text values, which could also be ordered
<u>datetime64</u>	Date and time values
<u>timedelta</u>	Difference between date time values
<u>NaN</u>	Nothing

It is important to add the correct data type to your data to optimise for both memory usage and cpu usage when performing operations like group by for example.

3. Sorting

Go to notebook

4. Notes

- in-memory pandas vs. databases uses
- Dask

<https://dask.org/>

• Cheat Sheet

Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
of rows in DataFrame.

`df['w'].nunique()`
of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

<code>sum()</code> Sum values of each object.	<code>min()</code> Minimum value in each object.
<code>count()</code> Count non-NA/null values of each object.	<code>max()</code> Maximum value in each object.
<code>median()</code> Median value of each object.	<code>mean()</code> Mean value of each object.
<code>quantile([0.25, 0.75])</code> Quantiles of each object.	<code>var()</code> Variance of each object.
<code>apply(function)</code> Apply function to each object.	<code>std()</code> Standard deviation of each object.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

<code>max(axis=1)</code> Element-wise max.	<code>min(axis=1)</code> Element-wise min.
<code>clip(lower=-10, upper=10)</code> Trim values at input thresholds	<code>abs()</code> Absolute value.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

<code>shift(1)</code> Copy with values shifted by 1.	<code>shift(-1)</code> Copy with values lagged by 1.
<code>rank(method='dense')</code> Ranks with no gaps.	<code>cumsum()</code> Cumulative sum.
<code>rank(method='min')</code> Ranks. Ties get min rank.	<code>cummax()</code> Cumulative max.
<code>rank(pct=True)</code> Ranks rescaled to interval [0, 1].	<code>cummin()</code> Cumulative min.
<code>rank(method='first')</code> Ranks. Ties go to first value.	<code>cumprod()</code> Cumulative product.

Group Data

`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group. Additional GroupBy functions:

<code>size()</code> Size of each group.	<code>agg(function)</code> Aggregate group using function.
--	---

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column

`df.plot.scatter(x='w', y='h')`
Scatter chart using pairs of points

Combine Data Sets

`adf` + `bdf` =

Standard Joins

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

`pd.merge(adf, bdf, how='outer', on='x1', indicator=True)`
Join data. Retain all values, all rows. Rows that appear in ydf but not zdf (Setdiff).

Filtering Joins

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

Set-like Operations

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

`pd.merge(ydf, zdf, how='outer', indicator=True)`
Rows that appear in either or both ydf and zdf (Union). Rows that appear in ydf but not zdf (Setdiff).

5. Exercises

Do the gap minder challenge and the results we expect should look something like this:

