Pandas for tabular data via its `DataFrame` object

A `DataFrame` is a 2-dimensional data structure typically consisting of rows that are observations and columns that are variables, like a spreadsheet. Get access to it via `import pandas as pd`. Each column is a *Series*, which is like a NumPy `ndarray`, and consists of values of the same `dtype` (typically one of `int, float, bool, str`).

Background: dictionary

Create a *dictionary* from a sequence of key:value pairs inside `{}`, e.g. `age = {'Anna': 6, 'Teresa': 9}`

Access via key, e.g. `age['Anna']`. A `DataFrame` is like a dictionary of column names.

Create, write, and read a `DataFrame`

- Use `pd.DataFrame(dict)` on a dictionary `dict`, e.g.
  ```
  df = pd.DataFrame({'item': ('milk', 'apples', 'bread'),
                     'price': (3.50, 0.40, 3.10),
                     'n': (2, 5, 1)})
  ```
- Write a file via `DataFrame.to_csv(path_or_buf, index=True)`; `index=False` omits index (row names); e.g. `df.to_csv(path_or_buf='groceries.csv', index=False)`
- Read from a ".csv" file via `pd.read_csv(filepath_or_buffer, names=None, index_col=None)`. Optional `names` provides column names. Optional `index_col`, a string or number, gives the column to use as row names ("index"); e.g.

  `groceries = pd.read_csv(filepath_or_buffer='groceries.csv')`

  `df = pd.read_csv('http://www.stat.wisc.edu/~jgillett/451/data/DJIA.csv', index_col='Symbol')`

Inspect

- `df.info()` summarizes the `DataFrame`
- `df.shape` gives dimensions as (#rows, #colums).
- `df.head(n=5)` gives first `n` rows, `df.tail(n=5)` gives last `n` rows
- `df.index` gives row names, `df.columns` gives column names
- `df.describe()` and `df.describe(include=[object])` summarize numeric and string columns
- `pd.crosstab(index, columns)` gives table of counts, e.g. `pd.crosstab(df['State'], df['Exchange'])`
- `df.min()`, `df.max()`, `df.idxmin()`, `df.idxmax()`, `df.mean()`, `df.std(ddof=1)`, e.g

  `df.max()`, `df['Price'].max()`, `df['MarketCap'].idxmax()`, `df['Price'].mean()`
- For a categorical variable `categorical`, `df.categorical.value_counts()` gives counts
- `df.groupby(by)` groups according to `by`, e.g. `df.groupby('Exchange').groups.keys()`, also `df.groupby('Exchange').size()`, `.sum()`, `.mean()`, `.std()`, `.min()`, `.max()`, `.describe()`

Select a subset

- Select a single `'column'` via `df.column` or `df['column']`, e.g. `df.Price`, `df['Price']`
- Select a list of columns in `column_names` via `df[column_names]`, e.g. `df[['Price', 'MarketCap']]`
- Select a row via `df.loc['row_name']`, e.g. `df.loc['AAPL']`
- Select a slice of rows from `low` to `high - 1` via `df[low:high]`, e.g. `df[0:3]`
- Select by position via `df.iloc[r, c]`, where each of `r` and `c` is an integer, a list of integers, or a `low:high` range, e.g. `df.iloc[0:3, [0, 2, 3]]`
- Select by name via `df.loc['index', 'colname']` or `df.loc[index_list, colname_list]` e.g. `df.loc['AAPL', 'State']` or `df.loc[['AAPL', 'MMM'], ['State', 'Industry']]`
- Select rows according to a logical `condition` via `df[condition]`, e.g. `df[df.Exchange == 'NASDAQ']`

Sort

- `df.sort_index(axis=0, ascending=True)` sorts rows (for `axis=0`) or columns (for `axis=1`) by name, e.g. `df.sort_index(axis=0, ascending=True) # also try 1, False`
- `df.sort_values(by, axis=0, ascending=True)` sorts according to the names in `by`, e.g.

```
df.sort_values(by=['Industry', 'State'], axis=0, ascending=True) # try [False, True]
```

Add to a `DataFrame`

- Add `new_column` via `df[new_column] = ...`, e.g. `groceries['weight'] = (8, 0.3, 1.0)`
- Add a `new_row` by concatenating a new `DataFrame`. e.g.

```
new_row = pd.DataFrame({'item':'popcorn', 'price':3, 'n':1, 'weight':2}, index=[0])
pd.concat([groceries, new_row], ignore_index=True) # ignore to reset second 0
```

Examples

- Find the average Price of NASDAQ stocks, e.g.

```
df[df.Exchange == 'NASDAQ'].Price.mean() # try NYSE
```

- Find the highest Price among the NASDAQ stocks with AvgVol under 10 million, e.g.

```
df[(df.Exchange == 'NASDAQ') & (df.AvgVol < 1e7)].Price.max()
```

- Find the average MarketCap of 3 highest-Price stocks, e.g.

```
df.sort_values(by='Price', ascending=False)[0:3].MarketCap.mean()
```

To learn more, see 10 minutes to pandas and Essential basic functionality.