Pandas

Instructor: Jorge Mendez
Logistics

- Homework 2 grades are out
- Homework 4 is out and due this Friday
- Homework 5 will be out this Friday and due the week after Spring Break
What is Pandas?

• Opensource data analysis library
• Main objects:
  • 2-dimensional DataFrame
  • 1-dimensional Series
• Built on NumPy
What is Pandas?

• Optimized for wide variety of data analysis operations
  • I/O to/from formatted files and data bases
  • Missing data handling
  • Slicing, indexing, reshaping, adding columns
  • Powerful grouping for aggregating and transforming data sets
  • Merging and joining data sets
  • Time-series functionality

• Applied in finance, neuroscience, economics, statistics, advertising, web analytics, and more.

• Virtually no modeling capabilities (easy to integrate with scikit-learn)
Series
Series

• One-dimensional array
• Possibly heterogeneous type (although usually not)
• Each element has a label referred to as index
• Missing values are represented as NaN
• May be MultiIndexed hierarchically
Series constructors

- **pd.Series(ndarray, index=None)** — series from array-like in same order
  - ndarray must be 1-dimensional
  - If index is provided, must be same length as ndarray
  - If index is not provided, will be 0, ..., len(ndarray) – 1

- **pd.Series(dict, index=None)** — series from dictionary
  - If index is provided, it gives the order over dict
  - If index contains keys not in dict, treated as missing value
  - If index does not contain some key in dict, it is discarded
  - If index is not provided, order will be insertion order into dict (for Python >= 3.6)

- **pd.Series(scalar, index)** — repeated scalar value
  - index is required
Live Example
Indexing and slicing Series

• Indices must be hashable types
• Index labels may not be unique, although that will raise errors in certain functions that require uniqueness
• Series objects can be indexed by either their index labels or their underlying 0-based index
• Slicing can also be done by either index type
  • Slicing based on index labels is done based on the order of the Series
Live Example
Series miscellaneous

• Series is array-like: valid argument for most NumPy functions
  • Array functions are modified to ignore missing values (NaN)
• Series is dict-like: get and set values by index label
• Series can be treated as arrays for vectorized operations
• Indices are automatically aligned: operating on two Series with different indices gives a Series with the union of the indices, where non-common indices are given NaN values
Live Example
DataFrames
DataFrame

• 2-dimensional labeled structure
• Possibly heterogeneous type (common across columns)
• Intuition: spreadsheet or SQL table
  • Each column is an attribute
  • Each row is a record
• Also: like a dictionary of Series objects
DataFrame constructors

• `pd.DataFrame(dict, index=None, columns=None)` — dict of Series or dicts
  • Keys from outer dict are columns, keys from inner dict are indices
  • If the keys in the outer dict are tuples, columns are MultiIndexed
  • `index` and `columns` treated like `index` for creating a Series from a dict
    • Dict key missing from index/columns: discarded
    • Order from index/columns
    • Index/columns missing from dict: treated as empty
DataFrame constructors

- `pd.DataFrame(dict, index=None, columns=None)` — dict of array-like
  - All arrays in dict must be the same length
  - If `index` is present, must be the same length as arrays
  - `columns` is treated same as before

- `pd.DataFrame(list, index=None, columns=None)` — list of dicts
  - Each dict is treated as a row
  - Column names are the union of the keys in all the dicts
Accessing DataFrame columns

- DataFrames can be indexed like dicts for accessing, adding, and deleting columns.
  
- Adding can be done with Series, array-like, or scalar:
  - `df[col] = Series` — Series with indices not in the DataFrame get those indices removed.
  - `df[col] = ndarray` — Array-like must have the same length as the indices in the DataFrame.
  - `df[col] = scalar` — Scalars are propagated to fill all indices.

- Columns are added at the end:
  - Use `insert()` to specify a different location.
Live Example
Indexing DataFrame

- `df[col]` — return a Series/DataFrame corresponding to the column(s) with key `col`
- `df.loc[idx, col]` — return a Series/DataFrame corresponding to the row(s) with index label(s) `idx` and column label(s) `col`
- `df.iloc[n_idx, n_col]` — return a Series/DataFrame corresponding to the row(s) with 0-based index(es) `n_idx` and column 0-based index(es) `n_col`
- `df[slice]` — return a DataFrame with all columns and rows sliced by `slice`
  - Slicing is like with Series, can be 0-indexed or label-indexed
- `df.at[idx, col], df.iat[idx, col]` — optimized versions of `loc, iloc` for accessing a scalar
Live Example
Boolean indexing of DataFrames

• df[bool_vec] — return a DataFrame with all columns and rows indexed by a Boolean array

• df[bool_df] — return DataFrame with original values where bool_df is True and NaN where it is False

• df.mask(bool_df) — return DataFrame with original values where bool_df is False and NaN where it is True

• df.duplicated(subset=None) — return Boolean DataFrame with True where a row is duplicated and False where it is not

• df.drop_duplicates(subset=None) — return DataFrame with original values but with duplicated rows removed (optional inplace argument)
Live Example
DataFrame miscellaneous

• DataFrames are aligned automatically both on columns and rows
  • Operations on misaligned DataFrames result in the union of columns and rows
• Series are broadcasted row-wise when operating with DataFrames
  • Exception: if the index is a date stamp, broadcasting is column-wise
• Scalar operations are elementwise
• DataFrames can be transposed with df.T
• NumPy functions can operate on DataFrames with numeric types
<table>
<thead>
<tr>
<th>Format Type</th>
<th>Data Description</th>
<th>Reader</th>
<th>Writer</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>CSV</td>
<td>read_csv</td>
<td>to_csv</td>
</tr>
<tr>
<td>text</td>
<td>JSON</td>
<td>read_json</td>
<td>to_json</td>
</tr>
<tr>
<td>text</td>
<td>HTML</td>
<td>read_html</td>
<td>to_html</td>
</tr>
<tr>
<td>text</td>
<td>Local clipboard</td>
<td>read_clipboard</td>
<td>to_clipboard</td>
</tr>
<tr>
<td>binary</td>
<td>MS Excel</td>
<td>read_excel</td>
<td>to_excel</td>
</tr>
<tr>
<td>binary</td>
<td>OpenDocument</td>
<td>read_excel</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>HDF5 Format</td>
<td>read_hdf</td>
<td>to_hdf</td>
</tr>
<tr>
<td>binary</td>
<td>Feather Format</td>
<td>read_feather</td>
<td>to_feather</td>
</tr>
<tr>
<td>binary</td>
<td>Parquet Format</td>
<td>read_parquet</td>
<td>to_parquet</td>
</tr>
<tr>
<td>binary</td>
<td>Msgpack</td>
<td>read_msgpack</td>
<td>to_msgpack</td>
</tr>
<tr>
<td>binary</td>
<td>Stata</td>
<td>read_stata</td>
<td>to_stata</td>
</tr>
<tr>
<td>binary</td>
<td>SAS</td>
<td>read_sas</td>
<td></td>
</tr>
<tr>
<td>binary</td>
<td>Python Pickle Format</td>
<td>read_pickle</td>
<td>to_pickle</td>
</tr>
<tr>
<td>SQL</td>
<td>SQL</td>
<td>read_sql</td>
<td>to_sql</td>
</tr>
<tr>
<td>SQL</td>
<td>Google Big Query</td>
<td>read_gbq</td>
<td>to_gbq</td>
</tr>
</tbody>
</table>
MultiIndexing

• Hierarchical indices for representing higher-dimensional data

• `pd.MultiIndex.from_tuples(list_of_tuples, names=None)` — first index in each tuple is highest level, last index is lowest level

• `pd.MultiIndex.from_product(list_of_lists, names=None)` — cartesian product: all possible combinations from lists

• `pd.MultiIndex.from_frame(DataFrame)` — each row is a multi-index, names from columns

• `index.levels` — unique indices from each level

• `index.get_level_values(level_idx)` — all indices in order at the level_idx level
Live Example
Combining DataFrames

• **pd.concat(list)** — concatenate list (or iterable) of DataFrames/Series
  • `axis=0`: 0 concatenate rows, 1 columns
  • `join='outer'`: ‘outer’ union over index, ‘inner’ intersection
  • `ignore_index=False`: whether to drop the index of concatenation axis. Useful if indices aren’t meaningful but may be repeated
  • `keys=None`: if present, create MultiIndex with keys at the outermost level (must be the length of list)
Combining DataFrames

• `pd.merge(left, right)` — implement SQL join operations on columns or indices
  • `how='inner'`
    • ‘inner’: SQL inner join, intersection of keys. Preserve order of left keys
    • ‘outer’: SQL outer join, union of keys. Sort keys lexicographically
    • ‘left’: SQL left outer join, only keys from left. Preserve order of left keys
    • ‘right’: SQL right outer join, only keys from right. Preserve order of right keys
  • on=None: which key to join on. If None, intersection of columns
  • `left_on/right_on=None`: which key from left/right to join on
  • `left_index/right_index=False`: use index from left/right as the join key
Live Example
Visualization

- Pandas integrates Matplotlib plotting functionality
- `df/s.plot()` — plot DataFrame or Series
  - If DataFrame, plot all rows as a separate Series with appropriate labeling
  - `kind`
    - ‘bar’ or ‘barh’ for bar plots
    - ‘hist’ for histogram
    - ‘box’ for boxplot
    - ‘kde’ or ‘density’ for density plots
    - ‘area’ for area plots
    - ‘scatter’ for scatter plots
    - ‘hexbin’ for hexagonal bin plots
    - ‘pie’ for pie plots
Takeaways

• Pandas is a powerful tool for data analysis
• Supports 1-D and 2-D data with Series and DataFrames
• Little support for higher dimensional data with MultiIndexing
• I/O to/from virtually any type of data file
• Integrated with Matplotlib for easy visualization
• Can be used as input to scikit-learn for modeling