COMP 333 Data Analytics

Python pandas

Greg Butler

Data Science Research Centre

and

Centre for Structural and Functional Genomics

and

Computer Science and Software Engineering
Concordia University, Montreal, Canada

gregb@cs.concordia.ca

Overview of Lecture

1. Python `array`
2. Python `numpy ndarray`
3. Python `pandas DataFrame`

Python Types

Built-in type	Operator	Mutable	Example	Description
`bool`		No	`True`	Boolean
`bytearray`		Yes	`bytearray(b'\x01\x04')`	Array of bytes
`bytes`	`b''`	No	`b'\x00\x17\x02'`	
`complex`		No	`(1+4j)`	Complex number
`dict`	`{:}`	Yes	`{'a': True, 45: 'b'}`	Dictionary, indexed by, e.g., strings
`float`		No	`3.1`	Floating point number
`frozenset`		No	`frozenset({1, 3, 4})`	Immutable set
`int`		No	`17`	Integer
`list`	`[]`	Yes	`[1, 3, 'a']`	List
`set`	`{}`	Yes	`{1, 2}`	Set with unique elements
`slice`	`:`	No	`slice(1, 10, 2)`	Slice indices
`str`	`""` or `''`	No	`"Hello"`	String
`tuple`	`(,)`	No	`(1, 'Hello')`	Tuple

Python `array`

Module `array` in Python 3.3

class array.array(typecode[, initializer])

array

an object type for an array of basic values:
    characters, integers, floating point numbers
Arrays behave very much like lists
    except the type of objects is constrained
The type is specified a type code, eg
'l' C signed long (int)
'u' unicode character
'd' C double (float)

Example

array('l')
array('l', [1, 2, 3, 4, 5])
array('d', [1.0, 2.0, 3.14])

Python Array Methods

Method	Description
append()	Adds an element at the end of the list
clear()	Removes all the elements from the list
copy()	Returns a copy of the list
count()	Returns the number of elements with the specified value
extend()	Add the elements of a list (or any iterable), to the end of the current list
index()	Returns the index of the first element with the specified value
insert()	Adds an element at the specified position
pop()	Removes the element at the specified position
remove()	Removes the first item with the specified value
reverse()	Reverses the order of the list
sort()	Sorts the list

Python numpy Types

Numpy type	Char	Mutable	Example	
`array`		Yes	`np.array([1, 2])`	One-, two, or many-dimensional
`matrix`		Yes	`np.matrix([[1, 2]])`	Two-dimensional matrix
`bool_`		—	`np.array([1], 'bool_')`	Boolean, one byte long
`int_`		—	`np.array([1])`	Default integer, same as C's long
`int8`	b	—	`np.array([1], 'b')`	8-bit signed integer
`int16`	h	—	`np.array([1], 'h')`	16-bit signed integer
`int32`	i	—	`np.array([1], 'i')`	32-bit signed integer
`int64`	l, p, q	—	`np.array([1], 'l')`	64-bit signed integer
`uint8`	B	—	`np.array([1], 'B')`	8-bit unsigned integer
`float_`		—	`np.array([1.])`	Default float
`float16`	e	—	`np.array([1], 'e')`	16-bit half precision floating point
`float32`	f	—	`np.array([1], 'f')`	32-bit precision floating point
`float64`	d	—		64-bit double precision floating point
`float128`	g	—	`np.array([1], 'g')`	128-bit floating point
`complex_`		—		Same as `complex128`
`complex64`		—		Single precision complex number
`complex128`		—	`np.array([1+1j])`	Double precision complex number
`complex256`		—		2 128-bit precision complex number

Python numpy ndarray

numpy.array(*object*, *dtype=None*, *copy=True*, *order='K'*, *subok=False*, *ndmin=0***)**

Create an array.

Parameters:

object : *array_like*

An array, any object exposing the array interface, an object whose __array__ method returns an array, or any (nested) sequence.

dtype : *data-type, optional*

The desired data-type for the array. If not given, then the type will be determined as the minimum type required to hold the objects in the sequence. This argument can only be used to 'upcast' the array. For downcasting, use the .astype(t) method.

copy : *bool, optional*

If true (default), then the object is copied. Otherwise, a copy will only be made if __array__ returns a copy, if obj is a nested sequence, or if a copy is needed to satisfy any of the other requirements (**dtype**, *order*, etc.).

Python numpy ndarray

Examples

```
>>> np.array([1, 2, 3])
array([1, 2, 3])
```

Upcasting:

```
>>> np.array([1, 2, 3.0])
array([ 1.,  2.,  3.])
```

More than one dimension:

```
>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])
```

Minimum dimensions 2:

```
>>> np.array([1, 2, 3], ndmin=2)
array([[1, 2, 3]])
```

Type provided:

```
>>> np.array([1, 2, 3], dtype=complex)
array([ 1.+0.j,  2.+0.j,  3.+0.j])
```

Python pandas Types

Pandas type	Mutable	Example	Description
`Series`	Yes	`pd.Series([2, 3, 6])`	One-dimension (vector-like)
`DataFrame`	Yes	`pd.DataFrame([[1, 2]])`	Two-dimensional (matrix-like)
`Panel`	Yes	`pd.Panel([[[1, 2]]])`	Three-dimensional (tensor-like)
`Panel4D`	Yes	`pd.Panel4D([[[[1]]]])`	Four-dimensional

Python pandas Types

Panel and Panel4D

are deprecated, and replaced by xarray

xarray

xarray: N-D labeled arrays and datasets in Python

xarray (formerly **xray**) is an open source project and Python package that makes working with labelled multi-dimensional arrays simple, efficient, and fun!

Xarray introduces labels in the form of dimensions, coordinates and attributes on top of raw NumPy-like arrays, which allows for a more intuitive, more concise, and less error-prone developer experience. The package includes a large and growing library of domain-agnostic functions for advanced analytics and visualization with these data structures.

Xarray was inspired by and borrows heavily from pandas, the popular data analysis package focused on labelled tabular data. It is particularly tailored to working with netCDF files, which were the source of xarray's data model, and integrates tightly with dask for parallel computing.

xarray.pydata.org

Python pandas DataFrame

pandas.DataFrame

class pandas.**DataFrame**(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*) [source]

Two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

Parameters:

data : *ndarray (structured or homogeneous), Iterable, dict, or DataFrame*
Dict can contain Series, arrays, constants, or list-like objects
Changed in version 0.23.0: If data is a dict, argument order is maintained for Python 3.6 and later.

index : *Index or array-like*
Index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided

columns : *Index or array-like*
Column labels to use for resulting frame. Will default to RangeIndex (0, 1, 2, …, n) if no column labels are provided

dtype : *dtype, default None*
Data type to force. Only a single dtype is allowed. If None, infer

copy : *boolean, default False*
Copy data from inputs. Only affects DataFrame / 2d ndarray input

Python pandas DataFrame

See also:

`DataFrame.from_records`

Constructor from tuples, also record arrays.

`DataFrame.from_dict`

From dicts of Series, arrays, or dicts.

`DataFrame.from_items`

From sequence of (key, value) pairs pandas.read_csv, pandas.read_table, pandas.read_clipboard.

Python pandas DataFrame

Attributes

`T`	Transpose index and columns.
`at`	Access a single value for a row/column label pair.
`axes`	Return a list representing the axes of the DataFrame.
`blocks`	(DEPRECATED) Internal property, property synonym for as_blocks().
`columns`	The column labels of the DataFrame.
`dtypes`	Return the dtypes in the DataFrame.
`empty`	Indicator whether DataFrame is empty.
`ftypes`	Return the ftypes (indication of sparse/dense and dtype) in DataFrame.
`iat`	Access a single value for a row/column pair by integer position.
`iloc`	Purely integer-location based indexing for selection by position.
`index`	The index (row labels) of the DataFrame.
`is_copy`	Return the copy.
`ix`	A primarily label-location based indexer, with integer position fallback.
`loc`	Access a group of rows and columns by label(s) or a boolean array.
`ndim`	Return an int representing the number of axes / array dimensions.
`shape`	Return a tuple representing the dimensionality of the DataFrame.
`size`	Return an int representing the number of elements in this object.
`style`	Property returning a Styler object containing methods for building a styled HTML representation fo the DataFrame.
`values`	Return a Numpy representation of the DataFrame.

Python pandas DataFrame

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
       index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
     [[4, 7, 10],
      [5, 8, 11],
      [6, 9, 12]],
     index=[1, 2, 3],
     columns=['a', 'b', 'c'])
```

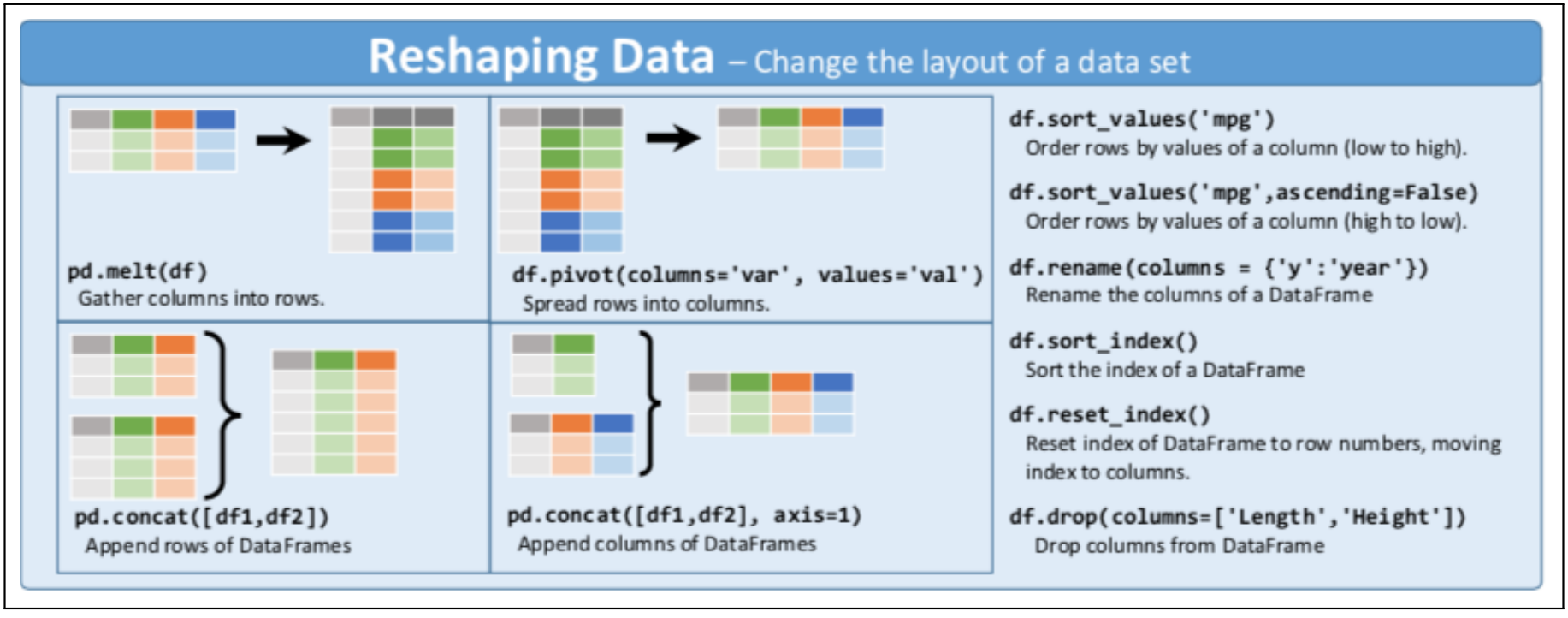
Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
           names=['n','v']))
```

Create DataFrame with a MultiIndex

Python pandas DataFrame



Python pandas DataFrame

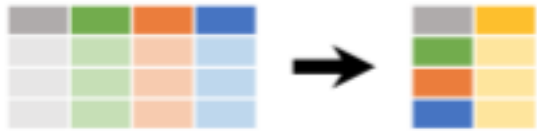
Summarize Data

`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:

`sum()`
Sum values of each object.

`count()`
Count non-NA/null values of each object.

`median()`
Median value of each object.

`quantile([0.25,0.75])`
Quantiles of each object.

`apply(function)`
Apply function to each object.

`min()`
Minimum value in each object.

`max()`
Maximum value in each object.

`mean()`
Mean value of each object.

`var()`
Variance of each object.

`std()`
Standard deviation of each object.

Python pandas DataFrame

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.

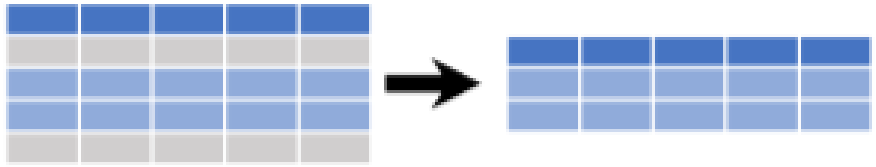
Additional GroupBy functions:

`size()`
Size of each group.

`agg(function)`
Aggregate group using function.

Python pandas DataFrame

Subset Observations (Rows)



`df[df.Length > 7]`
Extract rows that meet logical criteria.

`df.drop_duplicates()`
Remove duplicate rows (only considers columns).

`df.head(n)`
Select first n rows.

`df.tail(n)`
Select last n rows.

`df.sample(frac=0.5)`
Randomly select fraction of rows.

`df.sample(n=10)`
Randomly select n rows.

`df.iloc[10:20]`
Select rows by position.

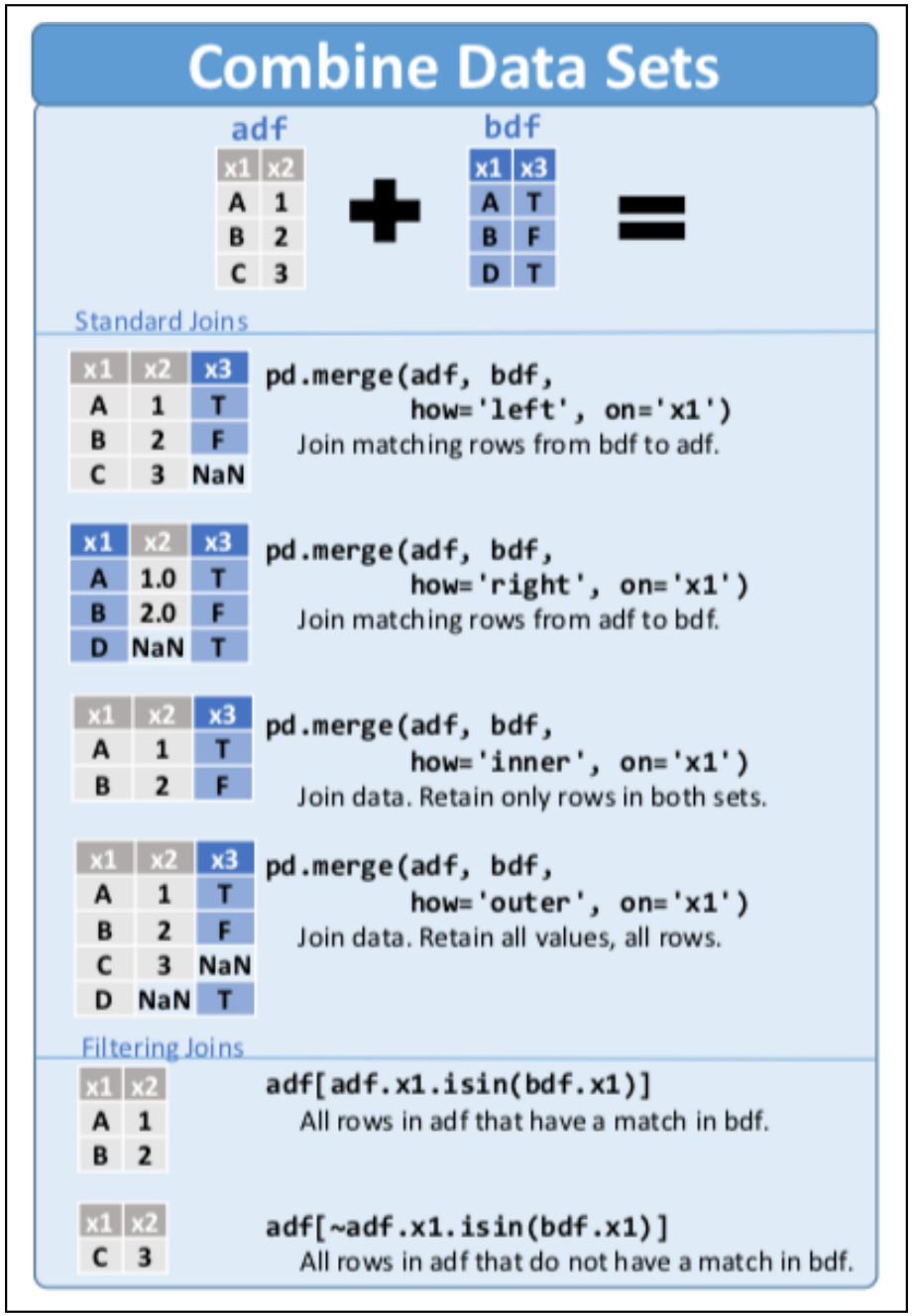
`df.nlargest(n, 'value')`
Select and order top n entries.

`df.nsmallest(n, 'value')`
Select and order bottom n entries.

Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	`df.column.isin(values)`	Group membership
==	Equals	`pd.isnull(obj)`	Is NaN
<=	Less than or equals	`pd.notnull(obj)`	Is not NaN
>=	Greater than or equals	`&,|,~,^,df.any(),df.all()`	Logical and, or, not, xor, any, all

Python pandas DataFrame

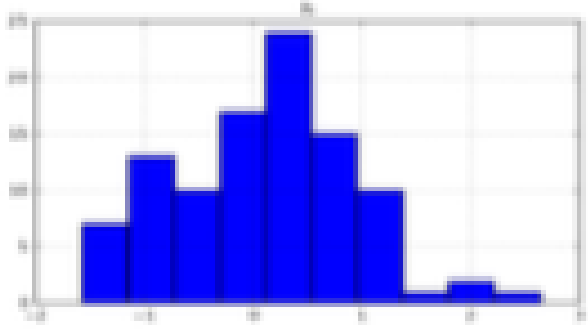


Python pandas DataFrame

Plotting

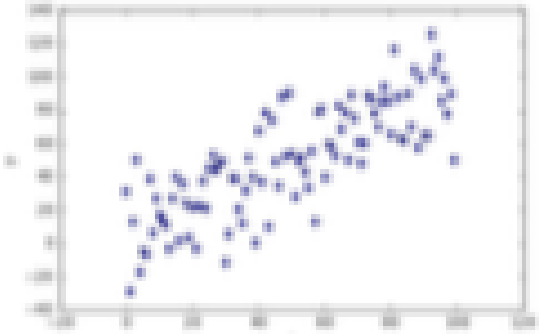
`df.plot.hist()`

Histogram for each column



`df.plot.scatter(x='w',y='h')`

Scatter chart using pairs of points



Python pandas DataFrame

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

```
df = (pd.melt(df)
        .rename(columns={
                    'variable' : 'var',
                    'value' : 'val'})
        .query('val >= 200')
    )
```