COMS BC1016
Introduction to Computational Thinking and Data Science

Lecture 8: Functions, Groups, Pivots, and Joins

Sept 29, 2025

Reminders

- HW 2 is due Wednesday
    - HW 3 due next week Wednesday
    - Remember to run the last cell
- Autograder is weird about Multiple Choice and True/False questions
    - TAs will manually grade

Midterm Info

- Midterm is during class Monday, October 20
  - Exam will cover material up until Wednesday, October 8
  - Monday, October 13 is a holiday (Indigenous Peoples' Day)
  - TAs will lead a midterm review during class Wednesday, October 15
- Paper exam
  - Can create and bring your own 5x7 notecard with notes on the exam
  - Notecard will be submitted along with the exam

Lecture Outline

- Functions
- Group and Pivot Tables
- `join`

Functions

Recall: Anatomy of a Function

Name, Parameters, Body, Return Statement

Example:

```
def convert_to_figs(weight):
    new_weight = (weight/7).round(1)
    return new_weight
```

Example

What does this function do?

- What type of input do you expect it takes?
- What type of output will it give?
- What's a reasonable name for the function?

```
def f(s):
    return s / sum(s) * 100
```

Example

What does this function do?

- What type of input do you expect it takes? Array
- What type of output will it give? Array
- What's a reasonable name for the function?

```
def f(s):
    return s / sum(s) * 100
```

Example

What does this function do?

- What type of input do you expect it takes? Array
- What type of output will it give? Array
- What's a reasonable name for the function? Anything related to percent

```
def percent(s):
    return s / sum(s) * 100
```

Function Documentation

```
sum?
```

```
Signature: sum(iterable, /, start=0)
Docstring:
Return the sum of a 'start' value (default: 0) plus an iterable of numbers

When the iterable is empty, return the start value.
This function is intended specifically for use with numeric values and may
reject non-numeric types.
Type:      builtin_function_or_method
```

Function Documentation

```
np.where?
```

```
Call signature:  np.where(*args, **kwargs)
Type:            _ArrayFunctionDispatcher
String form:     <built-in function where>
Docstring:
where(condition, [x, y], /)

Return elements chosen from `x` or `y` depending on `condition`.

.. note::
    When only `condition` is provided, this function is a shorthand for
    ``np.asarray(condition).nonzero()``. Using `nonzero` directly should be
    preferred, as it behaves correctly for subclasses. The rest of this
    documentation covers only the case where all three arguments are
    provided.

Parameters
----------
condition : array_like, bool
    Where True, yield `x`, otherwise yield `y`.
x, y : array_like
    Values from which to choose. `x`, `y` and `condition` need to be
    broadcastable to some shape.

Returns
-------
out : ndarray
    An array with elements from `x` where `condition` is True, and elements
    from `y` elsewhere.
```

Function Documentation

```
make_array?
```

```
Signature: make_array(*elements)
Docstring:
Returns an array containing all the arguments passed to this function.
A simple way to make an array with a few elements.

As with any array, all arguments should have the same type.

>>> make_array(0)
array([0])
>>> make_array(2, 3, 4)
array([2, 3, 4])
>>> make_array("foo", "bar")
array(['foo', 'bar'],
      dtype='<U3')
>>> make_array()
array([], dtype=float64)
File:      /opt/conda/lib/python3.12/site-packages/datascience/util.py
Type:      function
```

Adding Documentation

Putting a string in the first line of a function body defines the **Docstring**

- Typically describes behavior and expectations about its arguments

```
def convert_to_figs(weight):
    '''Divides the input by 7 (Figs weight) and
    then rounds to the first decimal place'''
    new_weight = (weight/7).round(1)
    return new_weight
```

Adding Documentation

Putting a string in the first line of a function body defines the **Docstring**

- Typically describes behavior and expectations about its arguments

```
def convert_to_figs(weight):
    '''Divides the input by 7 (Figs weight) and
    then rounds to the first decimal place'''
    new_weight = (weight/7).round(1)
    return new_weight
```

```
convert_to_figs?
```

```
Signature: convert_to_figs(weight)
Docstring:
Divides the input by 7 (Figs weight) and
then rounds to the first decimal place
File:      /tmp/ipykernel_201/903026817.py
Type:      function
```

Functions with Multiple Arguments/Parameters

Functions can take in multiple inputs

- Each argument is given a unique name and separated by commas

```
def convert_to_figs(weight, decimal_places):
    '''Divides the input by 7 (Figs weight) and then rounds to
    the given number of decimal places'''
    new_weight = (weight/7).round(decimal_places)
    return new_weight
```

Functions with Multiple Arguments/Parameters

Functions can take in multiple inputs

- Each argument is given a unique name and separated by commas
- Specifying default values for particular inputs to makes them optional

```
def convert_to_figs(weight, decimal_places=1):
    '''Divides the input by 7 (Figs weight) and then rounds to
    the given number of decimal places'''
    new_weight = (weight/7).round(decimal_places)
    return new_weight
```

Function Demo

Recall: `apply`

Use `apply` to call a function on each element in a column

```
def convert_to_figs(weight):
    new_weight = (weight/7).round(1)
    return new_weight
```

```
cat_tbl.apply(convert_to_figs, 'Weight')
```

Returns an array with convert_to_figs called on each element in the 'Weight' column

`apply` with Multiple Inputs

For functions with multiple inputs, `apply` can take multiple columns

```
def convert_to_figs(weight, decimal_places=1):
    new_weight = (weight/7).round(decimal_places)
    return new_weight


cat_tbl.apply(convert_to_figs, 'Weight', 'Precision')
```

Groups and Pivot Tables

Groups and Pivots

Two ways of **summarizing** table data by categorical variables

Recall: Prof Lee's Cat Census

Professor Lee is in a cat picture group chat. She has collected data on the cats shared in this chat:

Name	Age	Weight	Coloring	Sex	Owner
Ruby	14	8	tuxedo	F	Alice
Gertrude	15	12	tuxedo	F	Alice
Hamby	8	16	tabby	M	Bob
Fig	3	7	tabby	F	Bob
Corina	6	10	tortie	F	Carol
Frito	2	8.5	tabby	M	Carol

Grouping by a Single Column

The `group` method aggregates all rows with the same value in column `c`

- `tbl.group(c)`
- `tbl.group(c, func)`

`group` can optionally apply `func` to grouped values, for example:

- `len`: count of grouped values (default)
- `list`: list of all grouped values
- `sum`: total of all grouped values

```
cat_tbl.group('Owner')
```

Owner	count
Alice	2
Bob	2
Carol	2

```
cat_tbl.group('Owner', np.average)
```

Owner	Name average	Age average	Weight average	Coloring average	Sex average
Alice		14.5	10		
Bob		5.5	11.5		
Carol		4	9.25		

Grouping by Multiple Columns

The `group` method can also aggregate all rows that *share the combination of values* from multiple columns

```
cat_tbl.group(['Owner','Sex'])
```

Owner	Sex	count
Alice	F	2
Bob	F	1
Bob	M	1
Carol	F	1
Carol	M	1

```
cat_tbl.group(['Sex','Coloring'], sum)
```

Sex	Coloring	Name sum	Age sum	Weight sum	Owner sum
F	tabby		3	7	
F	tortie		6	10	
F	tuxedo		29	20	
M	tabby		10	24.5	

Pivot Tables

Cross-classifies according to *two* categorical variables

- Produces a grid of all possible combinations of the two categorical variables
- Grid entries are either counts or aggregated values

Create a pivot table where entries are counts:

```
tbl.pivot(col_var, row_var)
```

Create a pivot table where entries are aggregated according function `collect` on values in column `values`

```
tbl.pivot(col_var, row_var, values, collect)
```

Pivot Tables

```
tbl.pivot(col_var, row_var)
```

- `col_var`: Variable that forms column labels of grid
- `row_var`: Variable that forms row labels of grid

```
cat_tbl.pivot('Owner', 'Sex')
```

Sex	Alice	Bob	Carol
F	2	1	1
M	0	1	1

Pivot Tables

`tbl.pivot(col_var, row_var, values, collect)`

- `values`: Table column to aggregate

- `collect`: Function to aggregate with

Either include **both** `values` and `collect` or **neither**

```
cat_tbl.pivot('Owner', 'Sex', 'Age', np.average)
```

Sex	Alice	Bob	Carol
F	14.5	3	6
M	0	8	2

Group vs Pivot

Group

- One combo of grouping variables **per row**
- **Any number** of grouping variables
- Aggregate values of **all other columns** in the table
- Missing combos are **absent**

Pivot

- One combo of grouping variables **per entry**
- **Two** grouping variables: columns and rows
- Aggregate values of **values column**
- Missing combos **= 0 (or empty string)**

```
cat_tbl.group(['Sex','Coloring'], np.average)
```

Sex	Coloring	Name average	Age average	Weight average	Owner average
F	tabby		3	7	
F	tortie		6	10	
F	tuxedo		14.5	10	
M	tabby		5	12.25	

```
cat_tbl.pivot('Sex', 'Coloring', 'Weight', np.average)
```

Coloring	F	M
tabby	7	12.25
tortie	10	0
tuxedo	10	0

Joining Two Tables

Sometimes data about the same individuals are in different tables

- `join` combines the two datasets together

- Entries that do not appear in both tables are not included in the new table

To combine entries from `table1` and `table2` based on columns `c1` and `c2`

- `table1.join(c1, table2, c2)`

`join` Example

`bubble_teas`

cafe	drinks	prices
Gong Cha	Matcha Tea Latte	5.75
Tea Magic	Oolong Milk Tea	8
Hey Tea	Coconut Mango Boom	6.49
Moge Tee	Taro Milk Tea	7.45

`discounts`

% off	location
10	Gong Cha
25	Hey Tea
5	Moge Tee

`join` Example

`bubble_teas`

cafe	drinks	prices
Gong Cha	Matcha Tea Latte	5.75
Tea Magic	Oolong Milk Tea	8
Hey Tea	Coconut Mango Boom	6.49
Moge Tee	Taro Milk Tea	7.45

`discounts`

% off	location
10	Gong Cha
25	Hey Tea
5	Moge Tee

```
bubble_teas.join('cafe', discounts, 'location')
```

Match rows in this table…

…using values in this column …

…with rows in this second table…

…using values in this column.

`join` Example

`bubble_teas`

cafe	drinks	prices
Gong Cha	Matcha Tea Latte	5.75
Tea Magic	Oolong Milk Tea	8
Hey Tea	Coconut Mango Boom	6.49
Moge Tee	Taro Milk Tea	7.45

`discounts`

% off	location
10	Gong Cha
25	Hey Tea
5	Moge Tee

```
bubble_teas.join('cafe', discounts, 'location')
```

- Match rows in this table…
- …using values in this column …
- …with rows in this second table…
- …using values in this column.

output:

cafe	drinks	prices	% off
Gong Cha	Matcha Tea Latte	5.75	10
Hey Tea	Coconut Mango Boom	6.49	25
Moge Tee	Taro Milk Tea	7.45	5

`join` Example

`bubble_teas`

cafe	drinks	prices
Gong Cha	Matcha Tea Latte	5.75
Tea Magic	Oolong Milk Tea	8
Hey Tea	Coconut Mango Boom	6.49
Moge Tee	Taro Milk Tea	7.45

`discounts`

% off	location
10	Gong Cha
25	Hey Tea
5	Moge Tee

```
bubble_teas.join('cafe', discounts, 'location')
```

- `bubble_teas` — Match rows in this table…
- `'cafe'` — …using values in this column …
- `discounts` — …with rows in this second table…
- `'location'` — …using values in this column.

output:

cafe	drinks	prices	% off
Gong Cha	Matcha Tea Latte	5.75	10
Hey Tea	Coconut Mango Boom	6.49	25
Moge Tee	Taro Milk Tea	7.45	5

Next Class

- Conditionals and Iteration (if/else, for loops)