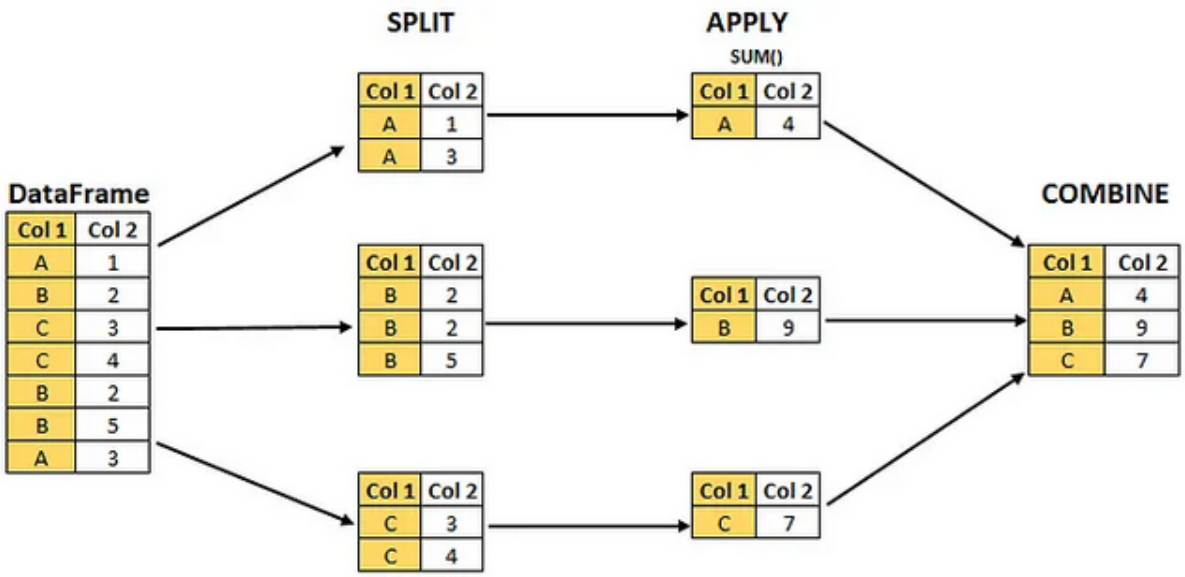
Announcements

HW#2 Due tonight at 11:59 PM
HW#3 Q3 – due 10/5 at 11:59 PM

Agenda Today

Lab 4 - Scatterplots
Lab 5 – Data Manipulation

Data Manipulation



1. **Split:** Split the data into groups based on some criteria thereby creating a GroupBy object. (We can use the column or a combination of columns to split the data into groups)
2. **Apply:** Apply a function to each group independently. (Aggregate, Transform, or Filter the data in this step)
3. **Combine:** Combine the results into a data structure (Pandas Series, Pandas DataFrame)

pandas.DataFrame.sample

DataFrame.sample(*n=None, frac=None, replace=False, weights=None, random_state=None, axis=None, ignore_index=False*) [source]

Return a random sample of items from an axis of object.

You can use *random_state* for reproducibility.

Parameters:

n : *int, optional*

Number of items from axis to return. Cannot be used with *frac*. Default = 1 if *frac* = None.

frac : *float, optional*

Fraction of axis items to return. Cannot be used with *n*.

replace : *bool, default False*

Allow or disallow sampling of the same row more than once.

weights : *str or ndarray-like, optional*

Default 'None' results in equal probability weighting. If passed a Series, will align with target object on index. Index values in weights not found in sampled object will be ignored and index values in sampled object not in weights will be assigned weights of zero. If called on a DataFrame, will accept the name of a column when axis = 0. Unless weights are a Series, weights must be same length as axis being sampled. If weights do not sum to 1, they will be normalized to sum to 1. Missing values in the weights column will be treated as zero. Infinite values not allowed.

random_state : *int, array-like, BitGenerator, np.random.RandomState, np.random.Generator, optional*

If int, array-like, or BitGenerator, seed for random number generator. If np.random.RandomState or np.random.Generator, use as given.

On this page
DataFrame.sample()
Show Source

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.sample html

pandas.DataFrame.assign

DataFrame.assign(kwargs)** [source]

Assign new columns to a DataFrame.

Returns a new object with all original columns in addition to new ones. Existing columns that are re-assigned will be overwritten.

Parameters:

****kwargs : *dict of {str: callable or Series}***

The column names are keywords. If the values are callable, they are computed on the DataFrame and assigned to the new columns. The callable must not change input DataFrame (though pandas doesn't check it). If the values are not callable, (e.g. a Series, scalar, or array), they are simply assigned.

Returns:

DataFrame

A new DataFrame with the new columns in addition to all the existing columns.

pandas.DataFrame.drop_duplicates

DataFrame.drop_duplicates(*subset=None, *, keep='first', inplace=False, ignore_index=False*) [source]

Return DataFrame with duplicate rows removed.

Considering certain columns is optional. Indexes, including time indexes are ignored.

Parameters:

subset : *column label or sequence of labels, optional*

Only consider certain columns for identifying duplicates, by default use all of the columns.

keep : *{'first', 'last', `False`}, default 'first'*

Determines which duplicates (if any) to keep.

- 'first' : Drop duplicates except for the first occurrence.
- 'last' : Drop duplicates except for the last occurrence.
- `False` : Drop all duplicates.

inplace : *bool, default* `False`

Whether to modify the DataFrame rather than creating a new one.

ignore_index : *bool, default* `False`

If `True`, the resulting axis will be labeled 0, 1, …, n - 1.

pandas.DataFrame.dropna

```
DataFrame.dropna(*, axis=0, how=_NoDefault.no_default,
thresh=_NoDefault.no_default, subset=None, inplace=False, ignore_index=False)
```

Remove missing values. [source]

See the User Guide for more on which values are considered missing, and how to work with missing data.

Parameters:

axis : *{0 or 'index', 1 or 'columns'}, default 0*

Determine if rows or columns which contain missing values are removed.

- 0, or 'index' : Drop rows which contain missing values.
- 1, or 'columns' : Drop columns which contain missing value.

Only a single axis is allowed.

how : *{'any', 'all'}, default 'any'*

Determine if row or column is removed from DataFrame, when we have at least one NA or all NA.

- 'any' : If any NA values are present, drop that row or column.
- 'all' : If all values are NA, drop that row or column.

thresh : *int, optional*

Require that many non-NA values. Cannot be combined with how.

subset : *column label or sequence of labels, optional*

Labels along other axis to consider, e.g. if you are dropping rows these would be a list of columns to include.

inplace : *bool, default False*

Operator	Description	Syntax
>	Greater than: True if the left operand is greater than the right	x > y
<	Less than: True if the left operand is less than the right	x < y
==	Equal to: True if both operands are equal	x == y
!=	Not equal to – True if operands are not equal	x != y
>=	Greater than or equal to: True if left operand is greater than or equal to the right	x >= y
<=	Less than or equal to: True if left operand is less than or equal to the right	x <= y

OPERATOR	NAME	DESCRIPTION	SYNTAX
&	Bitwise AND	Result bit 1,if both operand bits are 1;otherwise results bit 0.	x & y
|	Bitwise OR	Result bit 1,if any of the operand bit is 1; otherwise results bit 0.	x | y
~	Bitwise NOT	inverts individual bits	~x
^	Bitwise XOR	Results bit 1,if any of the operand bit is 1 but not both, otherwise results bit 0.	x ^ y

Warning/Possible Error

In [26]:
```
## Note that we can sort the full dataset "in place", which means that we don't
## have to assign the sorted dataframe to a new variable after it is sorted.

# To make sure that the orginial dataframe is sorted, we will use the 'in_place' argument
# as follows

sales04 = sales.loc[sales['Year'] == 2004, ]

sales04.sort_values(by = "Gross profit", ascending=False,inplace = True)

sales04.reset_index(inplace=True,drop=True) # note that we may need to reset the index of the dataframe

sales04.head()
```

```
C:\Users\f2per\AppData\Local\Temp\ipykernel_17356\96841036.py:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  sales04.sort_values(by = "Gross profit", ascending=False,inplace = True)
```

Built-in Functions

The Python interpreter has a number of functions and types built into it that are always available. They are listed here in alphabetical order.

Built-in Functions			
A abs() aiter() all() anext() any() ascii() **B** bin() bool() breakpoint() bytearray() bytes() **C** callable() chr() classmethod() compile() complex() **D** delattr() dict() dir() divmod()	**E** enumerate() eval() exec() **F** filter() float() format() frozenset() **G** getattr() globals() **H** hasattr() hash() help() hex() **I** id() input() int() isinstance() issubclass() iter()	**L** len() list() locals() **M** map() max() memoryview() min() **N** next() **O** object() oct() open() ord() **P** pow() print() property()	**R** range() repr() reversed() round() **S** set() setattr() slice() sorted() staticmethod() str() sum() super() **T** tuple() type() **V** vars() **Z** zip() **_** __import__()

pandas.DataFrame.apply

DataFrame.apply(*func, axis=0, raw=False, result_type=None, args=(), by_row='compat', **kwargs*) [source]

Apply a function along an axis of the DataFrame.

Objects passed to the function are Series objects whose index is either the DataFrame's index (`axis=0`) or the DataFrame's columns (`axis=1`). By default (`result_type=None`), the final return type is inferred from the return type of the applied function. Otherwise, it depends on the *result_type* argument.

Parameters:

func : *function*

Function to apply to each column or row.

axis : *{0 or 'index', 1 or 'columns'}, default 0*

Axis along which the function is applied:

- 0 or 'index': apply function to each column.
- 1 or 'columns': apply function to each row.

raw : *bool, default False*

Determines if row or column is passed as a Series or ndarray object:

- `False` : passes each row or column as a Series to the function.
- `True` : the passed function will receive ndarray objects instead. If you are just applying a NumPy reduction function this will achieve much better performance.

result_type : *{'expand', 'reduce', 'broadcast', None}, default None*

pandas.DataFrame.aggregate

`DataFrame.aggregate(func=None, axis=0, *args, **kwargs)` [source]

Aggregate using one or more operations over the specified axis.

Parameters:

func : *function, str, list or dict*

Function to use for aggregating the data. If a function, must either work when passed a DataFrame or when passed to DataFrame.apply.

Accepted combinations are:

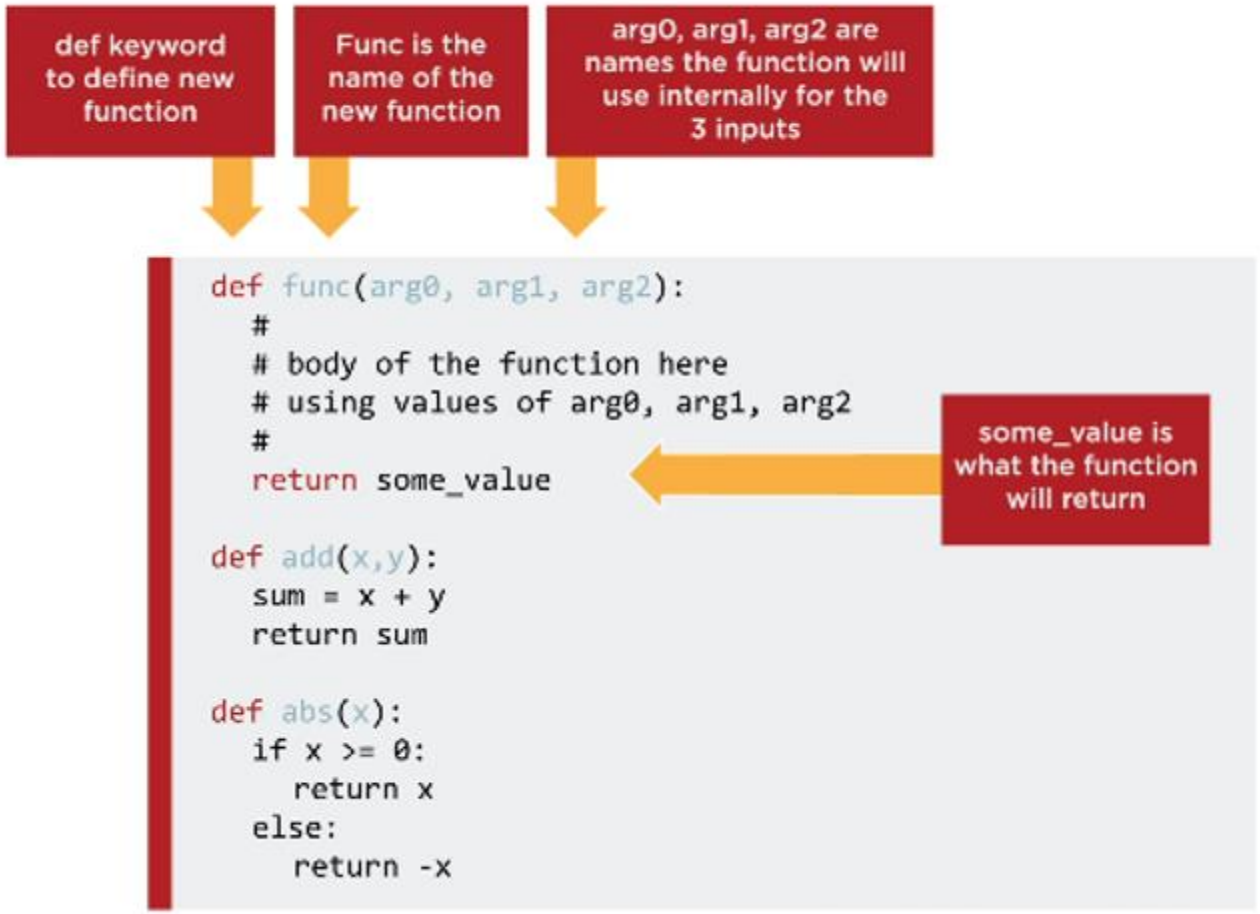
- function
- string function name
- list of functions and/or function names, e.g. `[np.sum, 'mean']`
- dict of axis labels -> functions, function names or list of such.

axis : *{0 or 'index', 1 or 'columns'}, default 0*

If 0 or 'index': apply function to each column. If 1 or 'columns': apply function to each row.

***args**

Defining a Function



Lambda functions

```
raise_to_power = lambda x, y: x ** y

raise_to_power(2, 3)
```

```
8
```

Anonymous functions

- Function map takes two arguments: `map(func, seq)`
- `map()` applies the function to ALL elements in the sequence

```
nums = [48, 6, 9, 21, 1]

square_all = map(lambda num: num ** 2, nums)

print(square_all)
```

```
<map object at 0x103e065c0>
```

Defining a function

```
def square():    # <- Function header
    new_value = 4 ** 2    # <- Function body
    print(new_value)
```

Return values from functions

- Return a value from a function using return

```
def square(value):
    new_value = value ** 2
    return new_value
num = square(4)

print(num)
```