Data Cleaning and Preparation

Part 2

Data Transformation

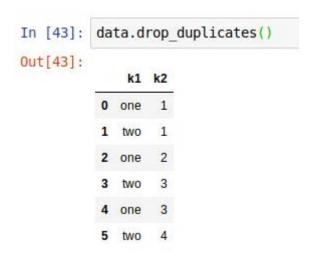
Part 1

Removing Duplicates

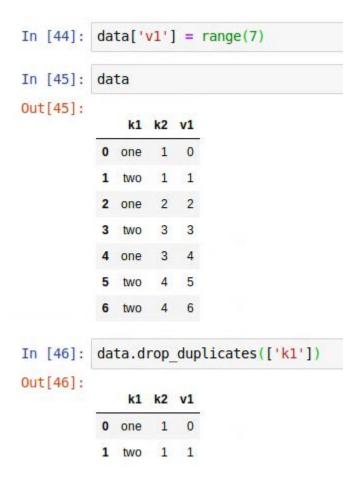
- Duplicate rows may be found in a DataFrame for any number of reasons.
- Here is an example:

 The DataFrame method duplicated returns a boolean Series indicating whether each row is a duplicate (has been observed in a previous row) or not:

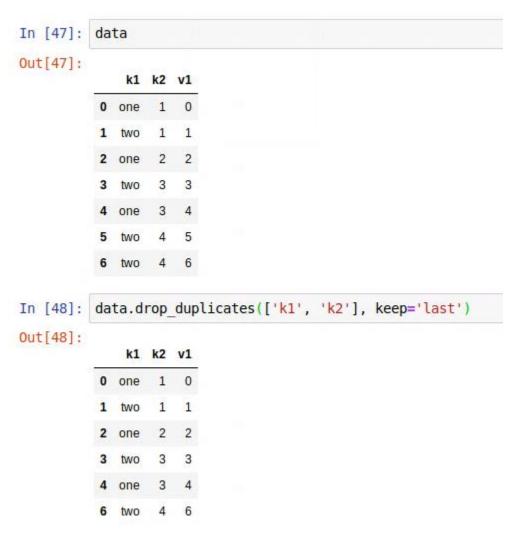
• Relatedly, drop_duplicates returns a DataFrame where the duplicated array is False:



- Both of these methods by default consider all of the columns; alternatively, you can specify any subset of them to detect duplicates.
- Suppose we had an additional column of values and wanted to filter duplicates only based on the ' k1 ' column:



- duplicated and drop_duplicates by default keep the first observed value combination.
- Passing keep='last' will return the last one:



Transforming Data Using a Function or Mapping

• For many datasets, you may wish to perform some transformation based on the values in an array, Series, or column in a DataFrame.

 Consider the following hypothetical data collected about various kinds of meat:

```
In [49]: data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
                                           'Pastrami', 'corned beef', 'Bacon',
                                           'pastrami', 'honey ham', 'nova lox'],
                                 'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})
In [50]: data
Out[50]:
                  food ounces
                           4.0
                 bacon
                           3.0
             pulled pork
                          12.0
                  bacon
                Pastrami
                           6.0
             corned beef
                           7.5
                 Bacon
                           8.0
                pastrami
                           3.0
           7 honey ham
                           5.0
                nova lox
                           6.0
```

- Suppose you wanted to add a column indicating the type of animal that each food came from.
- Let's write down a mapping of each distinct meat type to the kind of animal:

```
In [51]: meat_to_animal = {
    'bacon': 'pig',
    'pulled pork': 'pig',
    'pastrami': 'cow',
    'corned beef': 'cow',
    'honey ham': 'pig',
    'nova lox': 'salmon'
}
```

- The map method on a Series accepts a function or dict-like object containing a mapping, but here we have a small problem in that some of the meats are capitalized and others are not.
- Thus, we need to convert each value to lowercase using the str.lower Series method:

In [54]: data['animal'] = lowercased.map(meat_to_animal)

In [55]: data

Out[55]:

	food	ounces	animal
0	bacon	4.0	pig
1	pulled pork	3.0	pig
2	bacon	12.0	pig
3	Pastrami	6.0	cow
4	corned beef	7.5	cow
5	Bacon	8.0	pig
6	pastrami	3.0	cow
7	honey ham	5.0	pig
8	nova lox	6.0	salmon

We could also have passed a function that does all the work:

Replacing Values

- Filling in missing data with the fillna method is a special case of more general value replacement.
- As you've already seen, map can be used to modify a subset of values in an object but replace provides a simpler and more flexible way to do so.

- The -999 values might be sentinel values for missing data.
- To replace these with NA values that pandas understands, we can use replace, producing a new Series (unless you pass inplace=True):

• If you want to replace multiple values at once, you instead pass a list and then the substitute value:

• To use a different replacement for each value, pass a list of substitutes:

• The argument passed can also be a dict:

Renaming Axis Indexes

- Like values in a Series, axis labels can be similarly transformed by a function or mapping of some form to produce new, differently labeled objects.
- You can also modify the axes in-place without creating a new data structure.

• Like a Series, the axis indexes have a map method:

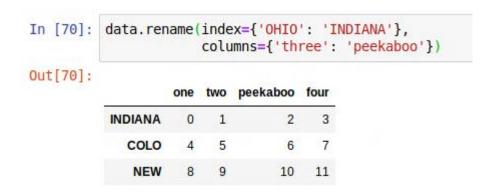
• You can assign to index, modifying the DataFrame in-place:



• If you want to create a transformed version of a dataset without modifying the original, a useful method is rename:



• Notably, rename can be used in conjunction with a dict-like object providing new values for a subset of the axis labels:



- rename saves you from the chore of copying the DataFrame manually and assigning to its index and columns attributes.
- Should you wish to modify a dataset in-place, pass inplace=True:

