UNIT-IV
DATA WRANGLING: CLEAN, TRANSFORM, MERGE, RESHAPE

Much of the programming work in data analysis and modeling is spent on data preparation: loading, cleaning, transforming, and rearranging.

Sometimes the way that data is stored in files or databases is not the way you need it for a data processing application.

Many people choose to do ad hoc processing of data from one form to another using a general purpose programming, like Python, Perl, R, or Java.....

Fortunately, pandas along with the Python standard library provide you with a high-level, flexible, and high-performance set of core manipulations and algorithms to enable you to wrangle data into the right form without much trouble.

Combining and Merging Data Sets:

Data contained in pandas objects can be combined together in a number of built-in ways:
• "pandas.merge" connects rows in DataFrames based on one or more keys. This will be familiar to users of SQL or other relational databases, as it implements database join operations.
• "pandas.concat" stacks together objects along an axis.
• "combine_first" instance method enables overlapping data to fill in missing values in one object with values from another.

Database-style DataFrame Merges:

- Merge or join operations combine data sets by linking rows using one or more keys.
- These operations are central to relational databases. The merge function in pandas is the main entry point for using these algorithms on your data.

```
1) import pandas as pd
df1=pd.DataFrame({'key':['a','b','a','a','b','c','b','d'],'d1':[11,22,33,44,55,66,77,88]})
df2=pd.DataFrame({'key':['a','a','b','c'],'d2':[1,2,3,4]})
print(df1)
print(df2)
pd.merge(df1, df2)
Output:
----------
key  d1
0   a  11
1   b  22
2   a  33
3   a  44
4   b  55
5   c  66
6   b  77
7   d  88
  key  d2
0   a   1
1   a   2
2   b   3
3   c   4
          key      d1       d2
0         a        11       1
1         a        11       2
2         a        33       1
3         a        33       2
4         a        44       1
```

```
5         a         44        2
6         b         22        3
7         b         55        3
8         b         77        3
9         c         66        4
```

2) pd.merge(df1, df2, on='key')

```
Output:
----------
          key       d1        d2
0         a         11        1
1         a         11        2
2         a         33        1
3         a         33        2
4         a         44        1
5         a         44        2
6         b         22        3
7         b         55        3
8         b         77        3
9         c         66        4
```

3) df1=pd.DataFrame({'lkey':['a','b','a','a','b','c','b','d'],'d1':[11,22,33,44,55,66,77,88]})
df2=pd.DataFrame({'rkey':['a','a','b','c'],'d2':[1,2,3,4]})
pd.merge(df1,df2,left_on='lkey', right_on='rkey')

```
Output:
----------
          lkey      d1        rkey      d2
0         a         11        a         1
1         a         11        a         2
2         a         33        a         1
3         a         33        a         2
4         a         44        a         1
5         a         44        a         2
6         b         22        b         3
7         b         55        b         3
8         b         77        b         3
9         c         66        c         4
```

4) df1=pd.DataFrame({'key':['a','b','a','a','b','c','b','d'],'d1':[11,22,33,44,55,66,77,88]})
df2=pd.DataFrame({'key':['a','a','b','c'],'d2':[1,2,3,4]})
pd.merge(df1,df2,how='outer')

```
Output:
----------
          key       d1        d2
0         a         11        1.0
1         a         11        2.0
2         a         33        1.0
3         a         33        2.0
4         a         44        1.0
5         a         44        2.0
6         b         22        3.0
7         b         55        3.0
8         b         77        3.0
9         c         66        4.0
10        d         88        NaN
```

5) To merge with multiple keys,
df1=pd.DataFrame({'key1':['a','b','c','a'],'key2':['x','x','y','z'],'d1':[11,22,33,44]})
df2=pd.DataFrame({'key1':['a','b','b'],'key2':['x','y','x'],'d2':[1,2,3]})
pd.merge(df1,df2,on=['key1','key2'],how='outer')

```
Output:
----------
        key1    key2    d1      d2
0       a       x       11.0    1.0
1       b       x       22.0    3.0
2       c       y       33.0    NaN
3       a       z       44.0    NaN
4       b       y       NaN     2.0
```

Merging on Index:

In some cases, the merge key or keys in a DataFrame will be found in its index. In this case, you can pass left_index=True or right_index=True (or both) to indicate that the index should be used as the merge key:

```
1) import pandas as pd
df1=pd.DataFrame({'key1':['a','b','c','a'],'key2':['x','x','y','z'],'d1':[11,22,33,44]})
df2=pd.DataFrame({'key1':['a','b','b'],'key2':['x','y','x'],'d2':[1,2,3]})
pd.merge(df1,df2,on=['key1','key2'],how='outer')
Output:
----------
        key1    key2    d1      d2
0       a       x       11.0    1.0
1       b       x       22.0    3.0
2       c       y       33.0    NaN
3       a       z       44.0    NaN
4       b       y       NaN     2.0
```

```
2) pd.merge(df1,df2,on=['key1','key2'],left_index=True, how='outer')
Output:
----------
        key1    key2    d1      d2
0       a       x       11.0    1.0
2       b       x       22.0    3.0
2       c       y       33.0    NaN
2       a       z       44.0    NaN
1       b       y       NaN     2.0
```

```
3) pd.merge(df1,df2,on=['key1','key2'],right_index=True, how='outer')
Output:
----------
        key1    key2    d1      d2
0       a       x       11.0    1.0
1       b       x       22.0    3.0
2       c       y       33.0    NaN
3       a       z       44.0    NaN
3       b       y       NaN     2.0
```

4) DataFrame has a more convenient **“join instance”** for merging by index. It can also be used to combine together many DataFrame objects having the same or similar indexes but non-overlapping columns.by using join()

```
df1=pd.DataFrame({'key1':['a','b','c','a'],'d1':[11,22,33,44]})
df2=pd.DataFrame({'key2':['a','b','b'],'d2':[1,2,3]})
df1.join(df2, how='left')
Output:
----------
        key1    d1      key2    d2
0       a       11      a       1.0
1       b       22      b       2.0
2       c       33      b       3.0
```

```
3         a         44        NaN       NaN
```

```
5) df1.join(df2, how='right')
Output:
----------
          key1      d1        key2      d2
0         a         11        a         1
1         b         22        b         2
2         c         33        b         3
```

```
6) df1=pd.DataFrame({'key1':['a','b','c','a'],'d1':[11,22,33,44]})
df2=pd.DataFrame({'key2':['a','b','b'],'d2':[1,2,3]})
df3=pd.DataFrame({'key3':['a','b','c'],'d3':[77,88,99]})
df1.join([df2,df3])
Output:
----------
          key1      d1        key2      d2        key3      d3
0         a         11        a         1.0       a         77.0
1         b         22        b         2.0       b         88.0
2         c         33        b         3.0       c         99.0
3         a         44        NaN       NaN       NaN       NaN
```

```
7) df1.join([df2,df3], how='outer')
Output:
----------
key1      d1        key2      d2        key3      d3
0         a         11        a         1.0       a         77.0
1         b         22        b         2.0       b         88.0
2         c         33        b         3.0       c         99.0
3         a         44        NaN       NaN       NaN       NaN
```

Concatenating along an Axis:

Another kind of data combination operation is alternatively referred to as concatenation, binding, or stacking. NumPy has a concatenate function for doing this with raw NumPy arrays:

```
1) import numpy as np
a=np.arange(12).reshape(3,4)
np.concatenate([a,a])
Output:
----------
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

```
2) import numpy as np
a=np.arange(12).reshape(3,4)
np.concatenate([a,a],axis=1)
Output:
----------
array([[ 0,  1,  2,  3,  0,  1,  2,  3],
       [ 4,  5,  6,  7,  4,  5,  6,  7],
       [ 8,  9, 10, 11,  8,  9, 10, 11]])
```

```
3) s1=pd.Series([1,2],index=['a','b'])
s2=pd.Series([3,4],index=['c','d'])
s3=pd.Series([5,6],index=['e','f'])
```

```
pd.concat([s1,s2,s3])
Output:
----------
a    1
b    2
c    3
d    4
e    5
f    6
dtype: int64
```

4) pd.concat([s1,s2,s3], axis=1, sort='False')

```
Output:
----------
        0       1       2
a       1.0     NaN     NaN
b       2.0     NaN     NaN
c       NaN     3.0     NaN
d       NaN     4.0     NaN
e       NaN     NaN     5.0
f       NaN     NaN     6.0
```

5)pd.concat([s1*100,s3])

```
Output:
----------
a    100
b    200
e      5
f      6
dtype: int64
```

6) pd.concat([s1,s2,s3],keys=['one','two','three'])

```
Output:
----------
one    a    1
       b    2
two    c    3
       d    4
three  e    5
       f    6
dtype: int64
```

7) pd.concat([s1,s2,s3],axis=1,keys=['one','two','three'])

```
Output:
----------
        one     two     three
a       1.0     NaN     NaN
b       2.0     NaN     NaN
c       NaN     3.0     NaN
d       NaN     4.0     NaN
e       NaN     NaN     5.0
f       NaN     NaN     6.0
```

8) The same logic extends to DataFrames also

```
df1=pd.DataFrame({'key1':['a','b','c','a'],'d1':[11,22,33,44]})
df2=pd.DataFrame({'key2':['a','b','b'],'d2':[1,2,3]})
pd.concat([df1,df2])
Output:
----------
        d1      d2      key1    key2
```

```
0       11.0    NaN     a       NaN
1       22.0    NaN     b       NaN
2       33.0    NaN     c       NaN
3       44.0    NaN     a       NaN
0       NaN     1.0     NaN     a
1       NaN     2.0     NaN     b
2       NaN     3.0     NaN     b
```

```
9) pd.concat([df1,df2],axis=1)
Output:
----------
        key1    d1      key2    d2
0       a       11      a       1.0
1       b       22      b       2.0
2       c       33      b       3.0
3       a       44      NaN     NaN
```

```
10) pd.concat([df1,df2], keys=['one','two'])
Output:
----------
        d1      d2      key1    key2
one     0       11.0    NaN     a       NaN
        1       22.0    NaN     b       NaN
        2       33.0    NaN     c       NaN
        3       44.0    NaN     a       NaN
two     0       NaN     1.0     NaN     a
        1       NaN     2.0     NaN     b
        2       NaN     3.0     NaN     b
```

Combining Date with Overlap:

Another data combination situation can’t be expressed as either a merge or concatenation operation. You may have two datasets whose indexes overlap in full or part.

```
1) import pandas as pd
import numpy as np
a=pd.Series([1,2,np.nan,4,np.nan,6,np.nan,8])
b=pd.Series([11,22,33,44,55,66,77,88])
np.where(pd.isnull(a),b,a)
Output:
----------
array([ 1.,  2., 33.,  4., 55.,  6., 77.,  8.])
```

```
2) df1=pd.DataFrame({'a':[1,2,np.nan,4],'b':[np.nan,6,np.nan,8]})
df2=pd.DataFrame({'a':[11,22,33,44],'b':[55,66,77,88]})
df1.combine_first(df2)
Output:
----------
        a       b
0       1.0     55.0
1       2.0     6.0
2       33.0    77.0
3       4.0     8.0
```

Reshaping and Pivoting:(Reshaping with hierarchical indexing)

--

There are a number of fundamental operations for rearranging tabular data. These are alternatingly referred to as reshape or pivot operations.

Hierarchical indexing provides a consistent way to rearrange data in a DataFrame. There are two primary actions:

- stack: this “rotates” or pivots from the columns in the data to the rows
- unstack: this pivots from the rows into the columns

```
1) data=pd.DataFrame(np.arange(6).reshape((2,3)), index=pd.Index(['cse','it'],name='branch'),
columns=pd.Index(['one','two','three'], name='number'))
print(data)
Output:
----------
number  one  two  three
branch
cse       0    1     2
it        3    4     5
```

```
2) r=data.stack()
print(r)
Output:
----------
branch  number
cse     one      0
        two      1
        three    2
it      one      3
        two      4
        three    5
dtype: int32
```

```
3) r=data.unstack()
print(r)
Output:
----------
number  branch
one     cse      0
          it     3
two     cse      1
          it     4
three   cse      2
          it     5
dtype: int32
```

4) Unstacking might introduce missing data if all of the values in the level aren't found

```
s1=pd.Series([1,2,3,4],index=['a','b','c','d'])
s2=pd.Series([5,6,7],index=['c','d','e'])
data=pd.concat([s1,s2], keys=['one','two'])
data.unstack()
Output:
----------
        a       b       c       d       e
one     1.0     2.0     3.0     4.0     NaN
two     NaN     NaN     5.0     6.0     7.0
```

```
5) s1=pd.Series([1,2,3,4],index=['a','b','c','d'])
s2=pd.Series([5,6,7],index=['c','d','e'])
data=pd.concat([s1,s2], keys=['one','two'])
data.unstack().stack()
Output:
----------
one  a    1.0
     b    2.0
```

```
       c    3.0
       d    4.0
two  c    5.0
       d    6.0
       e    7.0
dtype: float64
```

6) data.unstack().stack(dropna=False)

Output:

```
----------
one  a    1.0
     b    2.0
     c    3.0
     d    4.0
     e    NaN
two  a    NaN
     b    NaN
     c    5.0
     d    6.0
     e    7.0
dtype: float64
```

Data Transformation :(Removing Duplicates)

So far in this chapter we’ve been concerned with rearranging data. Filtering, cleaning, and other tranformations are another class of important operations.

Removing Duplicates:: Duplicate rows may be found in a DataFrame for any number of reasons.

```
1) d=pd.DataFrame({'k1':['a','b','a','c','b','d','b','a'],'k2':[1,2,1,4,2,2,2,1]})
print(d)
d.duplicated()
```

Output:

```
----------
  k1  k2
0  a   1
1  b   2
2  a   1
3  c   4
4  b   2
5  d   2
6  b   2
7  a   1
0    False
1    False
2     True
3    False
4     True
5    False
6     True
7     True
dtype: bool
```

2) d.drop_duplicates()

Output:

```
----------
        k1      k2
0       a       1
1       b       2
3       c       4
5       d       2
```

3) Suppose we had an additional column of values

```
d=pd.DataFrame({'k1':['a','b','a','c','b','d','b','a'],'k2':[1,2,1,4,2,2,2,1], 'v1':[1,2,1,4,2,2,2,2]})
d.drop_duplicates()
```

Output:

```
----------
        k1      k2      v1
0       a       1       1
1       b       2       2
3       c       4       4
5       d       2       2
7       a       1       2
```

4) And wanted to filter duplicates only based on 'k1'

```
d.drop_duplicates(['k1'])
```

Output:

```
----------
        k1      k2      v1
0       a       1       1
1       b       2       2
3       c       4       4
5       d       2       2
```

Replacing Values:

-----------------------

Filling in missing data with the fillna method can be thought of as a special case of more general value replacement. While map, as you’ve seen above, can be used to modify a subset of values in an object, replace provides a simpler and more flexible way to do so.

1) a=pd.Series([1,-99,3,-99,5,-99,7,-98])

```
print(a)
```

Output:

```
----------
0     1
1   -99
2     3
3   -99
4     5
5   -99
6     7
7   -98
dtype: int64
```

2) a.replace(-99,np.nan)

Output:

```
----------
0     1.0
1     NaN
2     3.0
3     NaN
4     5.0
5     NaN
6     7.0
7   -98.0
dtype: float64
```

3) a.replace([-99,-98],np.nan)

Output:

```
----------
0    1.0
```

```
1    NaN
2    3.0
3    NaN
4    5.0
5    NaN
6    7.0
7    NaN
dtype: float64
```

4) a.replace([-99,-98],77)

Output:

```
----------
0     1
1    77
2     3
3    77
4     5
5    77
6     7
7    77
dtype: int64
```

5) a.replace([-99,-98],[77,88])

Output:

```
----------
0     1
1    77
2     3
3    77
4     5
5    77
6     7
7    88
dtype: int64
```