CSC 223 - Advanced Scientific Programming

Pandas Combining Datasets

Combining Datasets

- In some data science tasks we need to combine data from different sources
- Pandas includes functions and methods for combining `Series` and `DataFrame` objects:
  - concatenation
  - database-style merges and joins

Concatenation

- The Pandas `concat` function can be used to concatenate `Series` or `DataFrame` objects
- The argument to the `concat` function is a sequence or mapping of `Series` or `DataFrame` objects

```
>>> s1 = pd.Series(['A', 'B', 'C'], index=[1,2,3])
>>> s2 = pd.Series(['D', 'E', 'F'], index=[4,5,6])
>>> pd.concat([s1, s2])
1    A
2    B
3    C
4    D
5    E
6    F
dtype: object
```

Concatenation

- The Pandas `concat` function has additional keyword arguments to specify additional concatenation options:
- Here is an (incomplete) list of some `concat` options
  - `axis`: the axis to concatenate along
  - `ignore_index`: if True, do not use the index values along the concatenation axis
  - `verify_integrity`: raise an exception if there are duplicate indices
  - `keys`: construct a hierarchical index using the keys values
  - `join`: specify an inner or outer join
  - `join_axes`: specific indices for joining

Concatenation Running Example

- Method for example DataFrame construction

```
>>> def make_df(cols, ind):
...       data = {c: [str(c) + str(i) for i in ind]
...               for c in cols}
...       return pd.DataFrame(data, ind)
...
>>> make_df('ABC', range(3))
    A   B   C
0  A0  B0  C0
1  A1  B1  C1
2  A2  B2  C2
```

Concatenation Running Example

- Concatenate default axis

```
>>> df1 = make_df('AB', [1, 2])
>>> df2 = make_df('AB', [3, 4])
>>> pd.concat([df1, df2])
    A   B
1  A1  B1
2  A2  B2
3  A3  B3
4  A4  B4
```

- Concatenate axis=1

```
>>> df3 = make_df('AB', [0, 1])
>>> df4 = make_df('CD', [0, 1])
>>> pd.concat([df3, df4], axis='col')
>>> pd.concat([df3, df4], axis=1)
    A   B   C   D
0  A0  B0  C0  D0
1  A1  B1  C1  D1
```

Concatenation Running Example

- Duplicate indices

```
>>> x = make_df('AB', [0,1])
>>> y = make_df('AB', [2, 3])
>>> y.index = x.index
>>> pd.concat([x, y])
    A   B
0  A0  B0
1  A1  B1
0  A2  B2
1  A3  B3
```

- Catching the repeats as an error

```
>>> pd.concat([x, y], verify_integrity=True)
ValueError: Indexes have overlapping values: Int64
```

Concatenation Running Example

- Ignoring the index

```
>>> pd.concat([x, y], ignore_index=True)
    A   B
0  A0  B0
1  A1  B1
2  A2  B2
3  A3  B3
```

- Adding MultiIndex keys

```
>>> pd.concat([x, y], keys=['x','y'])
      A   B
x 0  A0  B0
  1  A1  B1
y 0  A2  B2
  1  A3  B3
```

Concatenation Running Example

- Ignoring the index

```
>>> pd.concat([x, y], ignore_index=True)
    A   B
0  A0  B0
1  A1  B1
2  A2  B2
3  A3  B3
```

- Adding MultiIndex keys

```
>>> pd.concat([x, y], keys=['x','y'])
      A   B
x 0  A0  B0
  1  A1  B1
y 0  A2  B2
  1  A3  B3
```

Concatenation Running Example

- Different sets of column names (default outer join)

```
>>> df5 = make_df('ABC', [1, 2])
>>> df6 = make_df('BCD', [3, 4])
>>> pd.concat([df5, df6])
     A    B    C     D
1    A1   B1   C1   NaN
2    A2   B2   C2   NaN
3   NaN   B3   C3    D3
4   NaN   B4   C4    D4
```

- Different sets of column names (inner join)

```
>>> pd.concat([df5, df6], join='inner')
     B    C
1   B1   C1
2   B2   C2
3   B3   C3
4   B4   C4
```

Concatenation Running Example

- Specifying the column indices

```
>>> pd.concat([df5, df6], join_axes=[df5.columns])
     A    B    C
1    A1   B1   C1
2    A2   B2   C2
3   NaN   B3   C3
4   NaN   B4   C4
>>> pd.concat([df5, df6], join_axes=[df6.columns])
     B    C     D
1   B1   C1   NaN
2   B2   C2   NaN
3   B3   C3    D3
4   B4   C4    D4
```

Pandas `merge` Function

- Relational algebra is a formal set of rules for manipulating relational data
- The `merge` function is an interface to perform relational algebra join operations
- Categories of joins:
  - one-to-one
  - many-to-one
  - many-to-many

One-to-one Join Example

- A one-to-one join is similar to column-wise concatenation

```
>>> df1
    name group
0    Bob     A
1  Alice     B
2    Eve     B
>>> df2
    name  number
0    Eve       1
1  Alice       2
2    Bob       3
>>> pd.merge(df1, df2)
    name group  number
0    Bob     A       3
1  Alice     B       2
2    Eve     B       1
```

Many-to-one Join Example

- A many-to-one join is a join where one of the key columns contains duplicate entries

```
>>> df1
    name group  number
0    Bob     A       3
1  Alice     B       2
2    Eve     B       1
>>> df2
  group leader
0     A   Jack
1     B   Jill
>>> pd.merge(df1, df2)
    name group  number leader
0    Bob     A       3   Jack
1  Alice     B       2   Jill
2    Eve     B       1   Jill
```

Many-to-many Join Example

- A many-to-many join is a join where both the left and the right key columns contain duplicates

```
>>> df1
    name group  number
0    Bob     A       3
1  Alice     B       2
2    Eve     B       1
>>> df2
  group         skill
0     A          math
1     A   programming
2     B       biology
>>> pd.merge(df1, df2)
    name group  number         skill
0    Bob     A       3          math
1    Bob     A       3   programming
2  Alice     B       2       biology
3    Eve     B       1       biology
```

Pandas `merge` Function (continued)

- The `merge` function has additional keyword arguments
  - `on`: specify the key column
  - `left_on` and `right_on`: specify a key column that has a different name in each table
  - `left_index` and `right_index`: merge on index instead of column
  - `how`: specify an inner, outer, left, or right join
  - `suffixes`: append a suffix to conflicting column names