Combining

Combine multiple DataFrames through concatenation and merging.

Chapter Goals:

- Understand the methods used to combine DataFrame objects
- Write code for combining DataFrames

In the previous chapter, we discussed the `append` function for concatenating DataFrame rows. To concatenate multiple DataFrames along either rows or columns, we use the `pd.concat` `(https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html)` function.

The code below shows example usages of `pd.concat`.

```
df1 = pd.DataFrame({'c1':[1,2], 'c2':[3,4]},
                   index=['r1','r2'])
df2 = pd.DataFrame({'c1':[5,6], 'c2':[7,8]},
                   index=['r1','r2'])
df3 = pd.DataFrame({'c1':[5,6], 'c2':[7,8]})

concat = pd.concat([df1, df2], axis=1)
# Newline to separate print statements
print('{}\n'.format(concat))

concat = pd.concat([df2, df1, df3])
print('{}\n'.format(concat))

concat = pd.concat([df1, df3], axis=1)
print('{}\n'.format(concat))
```

Output

0.715s

```
0    5   7
1    6   8

     c1   c2   c1   c2
r1  1.0  3.0  NaN  NaN
r2  2.0  4.0  NaN  NaN
0   NaN  NaN  5.0  7.0
1   NaN  NaN  6.0  8.0
```

The `pd.concat` function takes in a list of pandas objects (normally a list of DataFrames) to concatenate. The function also takes in numerous keyword arguments, with `axis` being one of the more important ones. The `axis` argument specifies whether we concatenate the rows (`axis=0`, the default), or concatenate the columns (`axis=1`).

This works very similarly to concatenation in NumPy (https://www.educative.io/collection/page/6083138522447872/5629499534213120/5697982787747840/)

In the code example, the final call to `pd.concat` resulted in a DataFrame with many `NaN` values. This is because the row labels for `df1` and `df3` did not match, so result was padded with `NaN` in locations where values did not exist.

B. Merging

Apart from combining DataFrames through concatenation, we can also merge multiple DataFrames. The function we use is `pd.merge (http://pandas.pydata.org/pandas-docs/stable/generated/pandas.merge.html)`, which takes in two DataFrames for its two required arguments.

The code below shows how to use `pd.merge`.

```
mlb_df1 = pd.DataFrame({'name': ['john doe', 'al smith', 'sam black', 'john doe']
                        'pos': ['1B', 'C', 'P', '2B'],
                        'year': [2000, 2004, 2008, 2003]})
mlb_df2 = pd.DataFrame({'name': ['john doe', 'al smith', 'jack lee'],
                        'year': [2000, 2004, 2012],
                        'rbi': [80, 100, 12]})
```

```

print('{}\n'.format(mlb_df1))
print('{}\n'.format(mlb_df1))

mlb_merged = pd.merge(mlb_df1, mlb_df2)
print('{}\n'.format(mlb_merged))
```

Output 0.866s

```
        name pos  year
0   john doe  1B  2000
1   al smith   C  2004
2  sam black   P  2008
3   john doe  2B  2003

        name pos  year
0   john doe  1B  2000
1   al smith   C  2004
```

Without using any keyword arguments, `pd.merge` joins two DataFrames using all their common column labels. In the code example, the common labels between `mlb_df1` and `mlb_df2` were `name` and `year`.

The rows that contain the exact same values for the common column labels will be merged. Since `'john doe'` for year `2000` was in both `mlb_df1` and `mlb_df2`, its row was merged. However, `'john doe'` for year `2003` was only in `mlb_df1`, so its row was not merged.

The `pd.merge` function takes in many keyword arguments, but often none are needed to properly merge two DataFrames.

Time to Code!

The coding exercises for this chapter involve completing small functions that take in two DataFrame objects as input.

The first function, `concat_rows` will concatenate the rows of the two DataFrames.

Set `row_concat` equal to `pd.concat` with `[df1, df2]` as the only argument. Then return `row_concat`.

```
def concat_rows(df1, df2):
  row_concat = pd.concat([df1, df2])
  return row_concat
```

Show Results | Show Console

1 of 1 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓		0 1 2 0 1 5 3 1 -9 0 1 2 7 …	0 1 2 0 1 5 3 1 -9 0 1 2 7 …	Return value is correct, good job!

0.754s

The next function, `concat_cols` will concatenate the columns of the two input DataFrames.

Set `col_concat` equal to `pd.concat` with `[df1, df2]` as the required argument. Also set the `axis` keyword argument to `1`.

Then return `col_concat`.

```
def concat_cols(df1, df2):
  col_concat = pd.concat([df1, df2], axis=1)
  return col_concat


```

Show Results | Show Console

Show Results Show Console

1 of 1 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓		0 1 2 0 1 2 0 1 5 3 9 2 …	0 1 2 0 1 2 0 1 5 3 9 2 …	Return value is correct, good job!

0.878s

The final function, `merge_dfs` will merge the two input DataFrames along their columns.

Set `merged_df` equal to `pd.merge` with `df1` and `df2` as the first and second arguments, respectively.

Then return `merged_df`.

```
def merge_dfs(df1, df2):
  merged_df = pd.merge(df1, df2)
  return merged_df

```

Show Results Show Console

1 of 1 Tests Passed

Result	Input	Expected Output	Actual Output	Reason
✓		name pos year rbi 0 john doe …	name pos year rbi 0 john doe …	Return value is correct, good job!

0.861s

Back

DataFrame

Next

Indexing

✓ Mark as Completed