Combining DataFrames With Pandas

(https://github.com/learn-co-curriculum/dsc-combining-dataframes-pandas) (https://github.com/learn-co-curriculum/dsc-combining-dataframes-pandas/issues/new/choose)

Introduction

In this lesson, you'll learn how to combine DataFrames with concatenation. You'll also learn how to read in tables from SQL databases and store them in DataFrames, as well as the various types of joins that exist and how you can perform them in Pandas.

Objectives

You will be able to:

- Use concatenation to combine DataFrames
- Determine which type of join is preferred for two tables of data and a task
- Use different types of joins to merge dataframes

Concatenating DataFrames

Recall that "concatenation" means adding the contents of a second collection on to the end of the first collection. You learned how to do this when working with strings. For instance:

```
print('Data ' + 'Science!')
```

Since strings are a form of collections in Python, you can concatenate them as above.

DataFrames are also collections, so it stands to reason that pandas provides an easy way to concatenate them. Examine the following diagram from the pandas documentation on concatenation:

Help

df1

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

df2

	A	B	C	D
0	A4	B4	C4	D4
1	A5	B5	C5	D5
2	A6	B6	C6	D6
3	A7	B7	C7	D7

df3

	A	B	C	D
0	A8	B8	C8	D8
1	A9	B9	C9	D9
2	A10	B10	C10	D10
3	A11	B11	C11	D11

Result

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A3	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

In this example, three DataFrames have been concatenated, resulting in one larger DataFrame containing the contents in the order they were concatenated.

To perform a concatenation between two or more DataFrames, you pass in an array of the objects to concatenate to the `pd.concat()` function, as demonstrated below:

```
to_concat = [df1, df2, df3]
big_df = pd.concat(to_concat)
```

Help

Note that there are many different optional keyword arguments you can set with `pd.concat()` -- for a full breakdown of all the ways you can use this function, take a look at the pandas documentation (http://pandas.pydata.org/pandas-docs/stable/merging.html) .

Keys and Indexes

Every table in a database has a column that serves as the ***Primary Key***. In pandas, the index is the primary key for that table. You'll use these keys, along with the ***Foreign Key***, which points to a primary key value in another table, to execute ***Joins***. This allows us to "line up" information from multiple tables and combine them into one table. You'll learn more about primary keys and foreign keys in the future when you'll dive into SQL and relational databases, so don't worry too much about these concepts now. That said, you can use similar functionality in Pandas.

Often, it is useful for us to set a column to act as the index for a DataFrame. To do this, you would type:

```
some_dataframe.set_index('name_of_index_column', inplace=True)
```

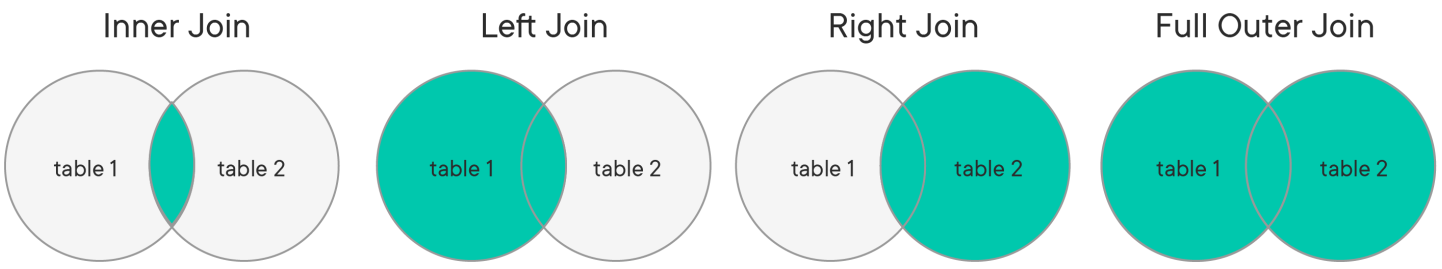
Note that this will mutate the dataset in place and set the column with the specified name as the index column of the DataFrame. If `inplace` is not specified it will default to False, meaning that a copy of the DataFrame with the requested changes will be returned, but the original object will remain unchanged.

NOTE: Running cells that make an `inplace` change more than once will often cause pandas to throw an error. If this happens, just restart the kernel.

By setting the index columns on DataFrames, you make it easy to join DataFrames later on. Note that this is not always feasible, but it's a useful step when possible.

Types of Joins

Joins are always executed between a ***Left Table*** and a ***Right Table***. There are four different types of joins you can execute. Consider the following Venn diagrams:



When thinking about joins, it is easy to conceptualize them as Venn diagrams.

- *in* returns all records from both tables
- *in* returns only the records with matching keys in both tables

Help

- A ***Left Join*** returns all the records from the left table, as well as any records from the right table that have a matching key with a record from the left table
- A ***Right Join*** returns all the records from the right table, as well as any records from the left table that have a matching key with a record from the right table

DataFrames contain a built-in `.join()` method. By default, the table calling the `.join()` method is always the left table. The following code snippet demonstrates how to execute a join in pandas:

```
joined_df = df1.join(df2, how='inner')
```

Note that to call `.join()` , you must pass in the right table. You can also set the type of join to perform with the `how` parameter. The options are `'left'` , `'right'` , `'inner'` , and `'outer'` .

If `how=` is not specified, it defaults to `'left'` .

NOTE: If both tables contain columns with the same name, the join will throw an error due to a naming collision, since the resulting table would have multiple columns with the same name. To solve this, pass in a value to `lsuffix=` or `rsuffix=` , which will append this suffix to the offending columns to resolve the naming collisions.

Summary

In this lesson, you learned how to use concatenation to join together multiple DataFrames in Pandas.

How do you feel about this lesson?

Have specific feedback?

[Tell us here! (https://github.com/learn-co-curriculum/dsc-combining-dataframes-pandas/issues/new/choose)](https://github.com/learn-co-curriculum/dsc-combining-dataframes-pandas/issues/new/choose)

Help