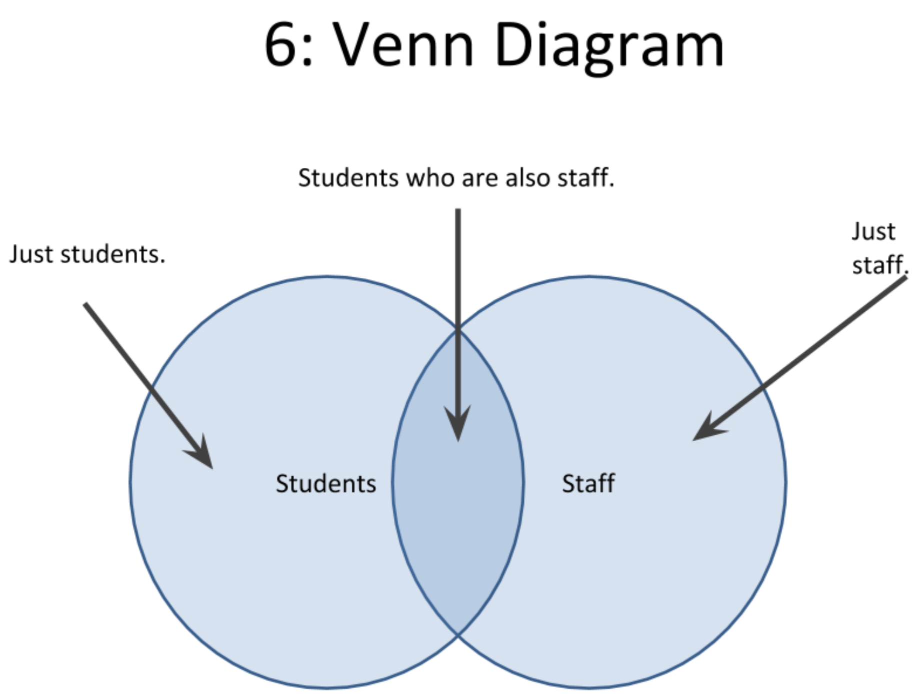
MergingDataFrame_ed

July 14, 2023

1 Merging

In this lecture we're going to address how you can bring multiple dataframe objects together, either by merging them horizontally, or by concatenating them vertically. Before we jump into the code, we need to address a little relational theory and to get some language conventions down. I'm going to bring in an image to help explain some concepts.

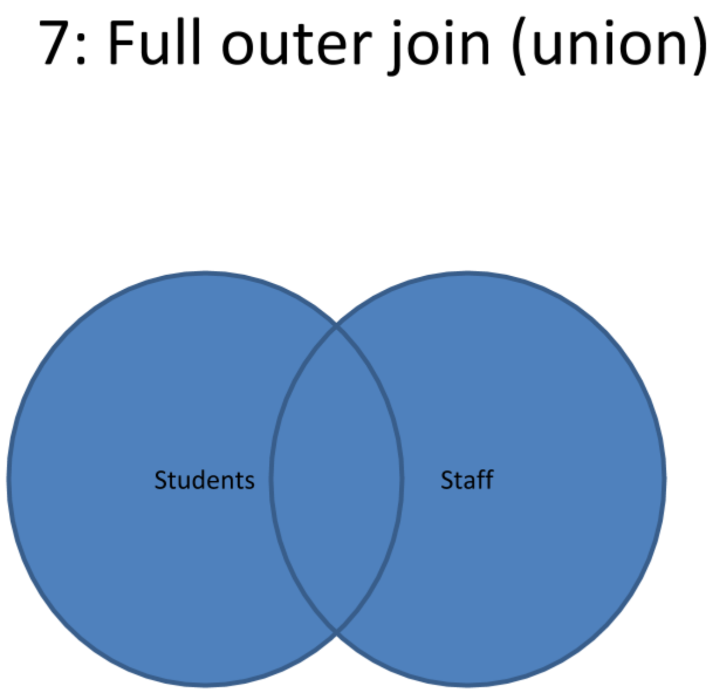


Ok, this is a Venn Diagram. A Venn Diagram is traditionally used to show set membership. For example, the circle on the left is the population of students at a university. The circle on the right is the population of staff at a university. And the overlapping region in the middle are all of those students who are also staff. Maybe these students run tutorials for a course, or grade assignments, or engage in running research experiments.

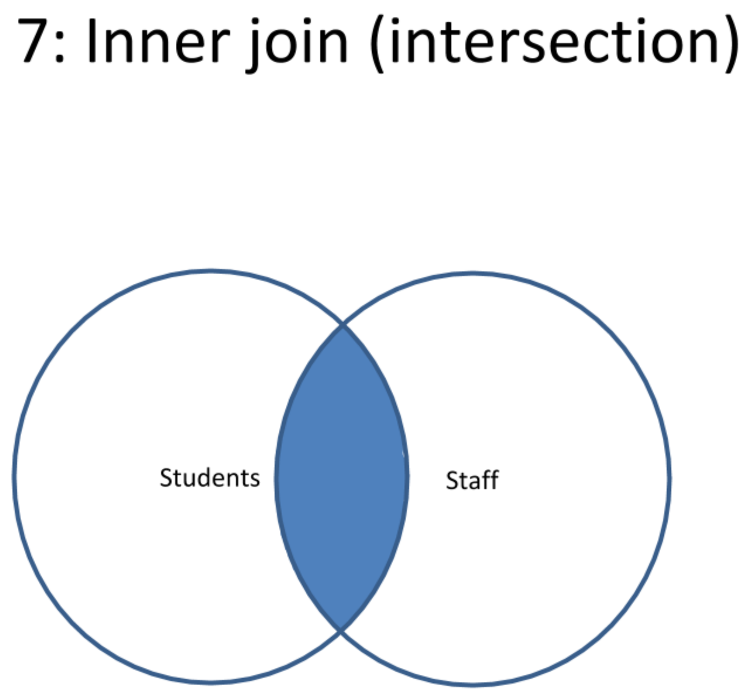
So, this diagram shows two populations whom we might have data about, but there is overlap between those populations.

When it comes to translating this to pandas, we can think of the case where we might have these two populations as indices in separate DataFrames, maybe with the label of Person Name. When we want to join the DataFrames together, we have some choices to make. First what if we want a list of all the people regardless of whether they're staff or student, and all of the information we can get on them? In database terminology, this is called a full outer join. And in set theory, it's called a union. In the Venn diagram, it represents everyone in any circle.

Here's an image of what that would look like in the Venn diagram.



It's quite possible though that we only want those people who we have maximum information for, those people who are both staff and students. Maybe being a staff member and a student involves getting a tuition waiver, and we want to calculate the cost of this. In database terminology, this is called an inner join. Or in set theory, the intersection. It is represented in the Venn diagram as the overlapping parts of each circle.



Here's what that looks like:

[1]:
```python
# With that background, let's see an example of how we would do this in pandas,␣
↪where we would use the merge
# function.
import pandas as pd

# First we create two DataFrames, staff and students.
staff_df = pd.DataFrame([{'Name': 'Kelly', 'Role': 'Director of HR'},
                         {'Name': 'Sally', 'Role': 'Course liasion'},
                         {'Name': 'James', 'Role': 'Grader'}])
# And lets index these staff by name
staff_df = staff_df.set_index('Name')
# Now we'll create a student dataframe
student_df = pd.DataFrame([{'Name': 'James', 'School': 'Business'},
                           {'Name': 'Mike', 'School': 'Law'},
                           {'Name': 'Sally', 'School': 'Engineering'}])
# And we'll index this by name too
student_df = student_df.set_index('Name')

# And lets just print out the dataframes
print(staff_df.head())
print(student_df.head())
```

```
                Role
```

```
Name
Kelly  Director of HR
Sally  Course liasion
James          Grader
             School
Name
James     Business
Mike           Law
Sally  Engineering
```

[2]:
```
# There's some overlap in these DataFrames in that James and Sally are both␣
↪students and staff, but Mike and
# Kelly are not. Importantly, both DataFrames are indexed along the value we␣
↪want to merge them on, which is
# called Name.
```

[3]:
```
# If we want the union of these, we would call merge() passing in the DataFrame␣
↪on the left and the DataFrame
# on the right and telling merge that we want it to use an outer join. We want␣
↪to use the left and right
# indices as the joining columns.

pd.merge(staff_df, student_df, how='outer', left_index=True, right_index=True)
```

[3]:
```
                 Role       School
Name
James          Grader     Business
Kelly  Director of HR          NaN
Mike              NaN          Law
Sally  Course liasion  Engineering
```

[4]:
```
# We see in the resulting DataFrame that everyone is listed. And since Mike␣
↪does not have a role, and John
# does not have a school, those cells are listed as missing values.

# If we wanted to get the intersection, that is, just those who are a student␣
↪AND a staff, we could set the
# how attribute to inner. Again, we set both left and right indices to be true␣
↪as the joining columns
pd.merge(staff_df, student_df, how='inner', left_index=True, right_index=True)
```

[4]:
```
                 Role       School
Name
Sally  Course liasion  Engineering
James          Grader     Business
```

[5]:

```python
# And we see the resulting DataFrame has only James and Sally in it. Now there
↪are two other common use cases
# when merging DataFrames, and both are examples of what we would call set
↪addition. The first is when we
# would want to get a list of all staff regardless of whether they were
↪students or not. But if they were
# students, we would want to get their student details as well. To do this we
↪would use a left join. It is
# important to note the order of dataframes in this function: the first
↪dataframe is the left dataframe and
# the second is the right

pd.merge(staff_df, student_df, how='left', left_index=True, right_index=True)
```

[5]:

```
                  Role       School
Name
Kelly  Director of HR          NaN
Sally  Course liasion  Engineering
James          Grader     Business
```

[6]:

```python
# You could probably guess what comes next. We want a list of all of the
↪students and their roles if they were
# also staff. To do this we would do a right join.
pd.merge(staff_df, student_df, how='right', left_index=True, right_index=True)
```

[6]:

```
                  Role       School
Name
James          Grader     Business
Mike              NaN          Law
Sally  Course liasion  Engineering
```

[7]:

```python
# We can also do it another way. The merge method has a couple of other
↪interesting parameters. First, you
# don't need to use indices to join on, you can use columns as well. Here's an
↪example. Here we have a
# parameter called "on", and we can assign a column that both dataframe has as
↪the joining column

# First, lets remove our index from both of our dataframes
staff_df = staff_df.reset_index()
student_df = student_df.reset_index()

# Now lets merge using the on parameter
pd.merge(staff_df, student_df, how='right', on='Name')
```

```
[7]:     Name            Role       School
     0  James          Grader     Business
     1   Mike             NaN          Law
     2  Sally  Course liasion  Engineering
```

```
[8]: # Using the "on" parameter instead of a the index is how I find myself using
     ↪merge() the most.
```

```
[9]: # So what happens when we have conflicts between the DataFrames? Let's take a
     ↪look by creating new staff and
     # student DataFrames that have a location information added to them.
     staff_df = pd.DataFrame([{'Name': 'Kelly', 'Role': 'Director of HR',
                               'Location': 'State Street'},
                              {'Name': 'Sally', 'Role': 'Course liasion',
                               'Location': 'Washington Avenue'},
                              {'Name': 'James', 'Role': 'Grader',
                               'Location': 'Washington Avenue'}])
     student_df = pd.DataFrame([{'Name': 'James', 'School': 'Business',
                                 'Location': '1024 Billiard Avenue'},
                                {'Name': 'Mike', 'School': 'Law',
                                 'Location': 'Fraternity House #22'},
                                {'Name': 'Sally', 'School': 'Engineering',
                                 'Location': '512 Wilson Crescent'}])

     # In the staff DataFrame, this is an office location where we can find the
     ↪staff person. And we can see the
     # Director of HR is on State Street, while the two students are on Washington
     ↪Avenue, and these locations just
     # happen to be right outside my window as I film this. But for the student
     ↪DataFrame, the location information
     # is actually their home address.

     # The merge function preserves this information, but appends an _x or _y to
     ↪help differentiate between which
     # index went with which column of data. The _x is always the left DataFrame
     ↪information, and the _y is always
     # the right DataFrame information.

     # Here, if we want all the staff information regardless of whether they were
     ↪students or not. But if they were
     # students, we would want to get their student details as well.Then we can do a
     ↪left join and on the column of
     # Name

     pd.merge(staff_df, student_df, how='left', on='Name')
```

```
[9]:     Name             Role          Location_x       School             Location_y
     0  Kelly  Director of HR        State Street          NaN                    NaN
     1  Sally  Course liasion  Washington Avenue  Engineering    512 Wilson Crescent
     2  James          Grader  Washington Avenue     Business   1024 Billiard Avenue
```

```
[10]: # From the output, we can see there are columns Location_x and Location_y.␣
       ↪Location_x refers to the Location
      # column in the left dataframe, which is staff dataframe and Location_y refers␣
       ↪to the Location column in the
      # right dataframe, which is student dataframe.

      # Before we leave merging of DataFrames, let's talk about multi-indexing and␣
       ↪multiple columns. It's quite
      # possible that the first name for students and staff might overlap, but the␣
       ↪last name might not. In this
      # case, we use a list of the multiple columns that should be used to join keys␣
       ↪from both dataframes on the on
      # parameter. Recall that the column name(s) assigned to the on parameter needs␣
       ↪to exist in both dataframes.

      # Here's an example with some new student and staff data
      staff_df = pd.DataFrame([{'First Name': 'Kelly', 'Last Name': 'Desjardins',
                                'Role': 'Director of HR'},
                               {'First Name': 'Sally', 'Last Name': 'Brooks',
                                'Role': 'Course liasion'},
                               {'First Name': 'James', 'Last Name': 'Wilde',
                                'Role': 'Grader'}])
      student_df = pd.DataFrame([{'First Name': 'James', 'Last Name': 'Hammond',
                                  'School': 'Business'},
                                 {'First Name': 'Mike', 'Last Name': 'Smith',
                                  'School': 'Law'},
                                 {'First Name': 'Sally', 'Last Name': 'Brooks',
                                  'School': 'Engineering'}])

      # As you see here, James Wilde and James Hammond don't match on both keys since␣
       ↪they have different last
      # names. So we would expect that an inner join doesn't include these␣
       ↪individuals in the output, and only Sally
      # Brooks will be retained.
      pd.merge(staff_df, student_df, how='inner', on=['First Name','Last Name'])
```

```
[10]:   First Name Last Name            Role       School
      0      Sally    Brooks  Course liasion  Engineering
```

```
[11]: # Joining dataframes through merging is incredibly common, and you'll need to␣
       ↪know how to pull data from
```

```
# different sources, clean it, and join it for analysis. This is a staple not␣
↪only of pandas, but of database
# technologies as well.
```

[12]:
```
# If we think of merging as joining "horizontally", meaning we join on similar␣
↪values in a column found in two
# dataframes then concatenating is joining "vertically", meaning we put␣
↪dataframes on top or at the bottom of
# each other

# Let's understand this from an example. You have a dataset that tracks some␣
↪information over the years. And
# each year's record is a separate CSV and every CSV ofr every year's record␣
↪has the exactly same columns.
# What happens if you want to put all the data, from all years' record,␣
↪together? You can concatenate them.
```

[13]:
```
# Let's take a look at the US Department of Education College Scorecard data It␣
↪has each US university's data
# on student completion, student debt, after-graduation income, etc. The data␣
↪is stored in separate CSV's with
# each CSV containing a year's record Let's say we want the records from 2011␣
↪to 2013 we first create three
# dataframe, each containing one year's record. And, because the csv files␣
↪we're working with are messy, I
# want to supress some of the jupyter warning messages and just tell read_csv␣
↪to ignore bad lines, so I'm
# going to start the cell with a cell magic called %%capture
```

[14]:
```
%%capture
df_2011 = pd.read_csv("datasets/college_scorecard/MERGED2011_12_PP.csv",␣
↪error_bad_lines=False)
df_2012 = pd.read_csv("datasets/college_scorecard/MERGED2012_13_PP.csv",␣
↪error_bad_lines=False)
df_2013 = pd.read_csv("datasets/college_scorecard/MERGED2013_14_PP.csv",␣
↪error_bad_lines=False)
```

[15]:
```
# Let's get a view of one of the dataframes
df_2011.head(3)
```

[15]:
```
     UNITID      OPEID OPEID6                               INSTNM  \
0  100654.0   100200.0   1002             Alabama A & M University
1  100663.0   105200.0   1052  University of Alabama at Birmingham
2  100690.0  2503400.0  25034                   Amridge University

         CITY STABBR        ZIP  ACCREDAGENCY INSTURL NPCURL  …  \
```

```
0      Normal     AL       35762           NaN      NaN     NaN  …
1  Birmingham     AL  35294-0110           NaN      NaN     NaN  …
2  Montgomery     AL  36117-3553           NaN      NaN     NaN  …

  OMAWDP8_NOTFIRSTTIME_POOLED_SUPP OMENRUP_NOTFIRSTTIME_POOLED_SUPP  \
0                              NaN                              NaN
1                              NaN                              NaN
2                              NaN                              NaN

  OMENRYP_FULLTIME_POOLED_SUPP OMENRAP_FULLTIME_POOLED_SUPP  \
0                          NaN                          NaN
1                          NaN                          NaN
2                          NaN                          NaN

  OMAWDP8_FULLTIME_POOLED_SUPP OMENRUP_FULLTIME_POOLED_SUPP  \
0                          NaN                          NaN
1                          NaN                          NaN
2                          NaN                          NaN

  OMENRYP_PARTTIME_POOLED_SUPP OMENRAP_PARTTIME_POOLED_SUPP  \
0                          NaN                          NaN
1                          NaN                          NaN
2                          NaN                          NaN

  OMAWDP8_PARTTIME_POOLED_SUPP OMENRUP_PARTTIME_POOLED_SUPP
0                          NaN                          NaN
1                          NaN                          NaN
2                          NaN                          NaN

[3 rows x 1977 columns]
```

[16]:

```python
# We see that there is a whopping number of columns - more than 1900! We can calculate the length of each
# dataframe as well
print(len(df_2011))
print(len(df_2012))
print(len(df_2013))
```

```
15235
7793
7804
```

[17]:

```python
# That's a bit surprising that the number of schools in the scorecard for 2011 is almost double that of the
# next two years. But let's not worry about that. Instead, let's just put all three dataframes in a list and
```

```
# call that list frames and pass the list into the concat() function Let's see
↪what it looks like

frames = [df_2011, df_2012, df_2013]
pd.concat(frames)
```

[17]:

```
          UNITID      OPEID OPEID6  \
0       100654.0   100200.0   1002
1       100663.0   105200.0   1052
2       100690.0  2503400.0  25034
3       100706.0   105500.0   1055
4       100724.0   100500.0   1005
…            …          …    …
7799  48285703.0   157107.0   1571
7800  48285704.0   157101.0   1571
7801  48285705.0   157105.0   1571
7802  48285706.0   157100.0   1571
7803  48285707.0   157103.0   1571

                                           INSTNM            CITY STABBR  \
0                        Alabama A & M University          Normal     AL
1             University of Alabama at Birmingham      Birmingham     AL
2                              Amridge University      Montgomery     AL
3             University of Alabama in Huntsville      Huntsville     AL
4                        Alabama State University      Montgomery     AL
…                                             …             …    …
7799       Georgia Military College-Columbus Campus        Columbus     GA
7800       Georgia Military College-Valdosta Campus        Valdosta     GA
7801  Georgia Military College-Warner Robins Campus   Warner Robins     GA
7802                Georgia Military College-Online    Milledgeville     GA
7803        Georgia Military College-Stone Mountain  Stone Mountain     GA

             ZIP  ACCREDAGENCY INSTURL NPCURL  …  \
0          35762           NaN     NaN    NaN  …
1     35294-0110           NaN     NaN    NaN  …
2     36117-3553           NaN     NaN    NaN  …
3          35899           NaN     NaN    NaN  …
4     36104-0271           NaN     NaN    NaN  …
…            …           …     …    …   …
7799       31909           NaN     NaN    NaN  …
7800       31605           NaN     NaN    NaN  …
7801       31093           NaN     NaN    NaN  …
7802       31061           NaN     NaN    NaN  …
7803       30083           NaN     NaN    NaN  …

     OMAWDP8_NOTFIRSTTIME_POOLED_SUPP OMENRUP_NOTFIRSTTIME_POOLED_SUPP  \
0                                 NaN                              NaN
```

```
1                                      NaN                                   NaN
2                                      NaN                                   NaN
3                                      NaN                                   NaN
4                                      NaN                                   NaN
...                                 ...                               ...
7799                                   NaN                                   NaN
7800                                   NaN                                   NaN
7801                                   NaN                                   NaN
7802                                   NaN                                   NaN
7803                                   NaN                                   NaN

      OMENRYP_FULLTIME_POOLED_SUPP OMENRAP_FULLTIME_POOLED_SUPP  \
0                              NaN                          NaN
1                              NaN                          NaN
2                              NaN                          NaN
3                              NaN                          NaN
4                              NaN                          NaN
...                            ...                          ...
7799                           NaN                          NaN
7800                           NaN                          NaN
7801                           NaN                          NaN
7802                           NaN                          NaN
7803                           NaN                          NaN

      OMAWDP8_FULLTIME_POOLED_SUPP OMENRUP_FULLTIME_POOLED_SUPP  \
0                              NaN                          NaN
1                              NaN                          NaN
2                              NaN                          NaN
3                              NaN                          NaN
4                              NaN                          NaN
...                            ...                          ...
7799                           NaN                          NaN
7800                           NaN                          NaN
7801                           NaN                          NaN
7802                           NaN                          NaN
7803                           NaN                          NaN

      OMENRYP_PARTTIME_POOLED_SUPP OMENRAP_PARTTIME_POOLED_SUPP  \
0                              NaN                          NaN
1                              NaN                          NaN
2                              NaN                          NaN
3                              NaN                          NaN
4                              NaN                          NaN
...                            ...                          ...
7799                           NaN                          NaN
7800                           NaN                          NaN
7801                           NaN                          NaN
```

```
7802                         NaN                         NaN
7803                         NaN                         NaN

     OMAWDP8_PARTTIME_POOLED_SUPP OMENRUP_PARTTIME_POOLED_SUPP
0                            NaN                         NaN
1                            NaN                         NaN
2                            NaN                         NaN
3                            NaN                         NaN
4                            NaN                         NaN
…                              …                           …
7799                         NaN                         NaN
7800                         NaN                         NaN
7801                         NaN                         NaN
7802                         NaN                         NaN
7803                         NaN                         NaN

[30832 rows x 1977 columns]
```

[18]:
```
# As you can see, we have more observations in one dataframe and columns remain␣
↪the same. If we scroll down to
# the bottom of the output, we see that there are a total of 30,832 rows after␣
↪concatenating three dataframes.
# Let's add the number of rows of the three dataframes and see if the two␣
↪numbers match
len(df_2011)+len(df_2012)+len(df_2013)
```

[18]: 30832

[19]:
```
# The two numbers match! Which means our concatenation is successful. But wait,␣
↪now that all the data is
# concatenated together, we don't know what observations are from what year␣
↪anymore! Actually the concat
# function has a parameter that solves such problem with the keys parameter, we␣
↪can set an extra level of
# indices, we pass in a list of keys that we want to correspond to the␣
↪dataframes into the keys parameter

# Now let's try it out
pd.concat(frames, keys=['2011','2012','2013'])
```

[19]:
```
                 UNITID      OPEID OPEID6  \
2011 0         100654.0   100200.0   1002
     1         100663.0   105200.0   1052
     2         100690.0  2503400.0  25034
     3         100706.0   105500.0   1055
     4         100724.0   100500.0   1005
```

```
…                 …        …    …
2013 7799  48285703.0   157107.0   1571
     7800  48285704.0   157101.0   1571
     7801  48285705.0   157105.0   1571
     7802  48285706.0   157100.0   1571
     7803  48285707.0   157103.0   1571

                                                   INSTNM           CITY  \
2011 0                           Alabama A & M University         Normal
     1                University of Alabama at Birmingham     Birmingham
     2                                 Amridge University     Montgomery
     3                University of Alabama in Huntsville     Huntsville
     4                           Alabama State University     Montgomery
…                                                       …              …
2013 7799       Georgia Military College-Columbus Campus        Columbus
     7800       Georgia Military College-Valdosta Campus        Valdosta
     7801  Georgia Military College-Warner Robins Campus   Warner Robins
     7802                Georgia Military College-Online   Milledgeville
     7803         Georgia Military College-Stone Mountain  Stone Mountain

          STABBR         ZIP  ACCREDAGENCY INSTURL NPCURL  …  \
2011 0        AL       35762           NaN     NaN    NaN  …
     1        AL  35294-0110           NaN     NaN    NaN  …
     2        AL  36117-3553           NaN     NaN    NaN  …
     3        AL       35899           NaN     NaN    NaN  …
     4        AL  36104-0271           NaN     NaN    NaN  …
…            …           …            …       …      …  …
2013 7799     GA       31909           NaN     NaN    NaN  …
     7800     GA       31605           NaN     NaN    NaN  …
     7801     GA       31093           NaN     NaN    NaN  …
     7802     GA       31061           NaN     NaN    NaN  …
     7803     GA       30083           NaN     NaN    NaN  …

          OMAWDP8_NOTFIRSTTIME_POOLED_SUPP OMENRUP_NOTFIRSTTIME_POOLED_SUPP  \
2011 0                                 NaN                              NaN
     1                                 NaN                              NaN
     2                                 NaN                              NaN
     3                                 NaN                              NaN
     4                                 NaN                              NaN
…                                    …                              …
2013 7799                              NaN                              NaN
     7800                              NaN                              NaN
     7801                              NaN                              NaN
     7802                              NaN                              NaN
     7803                              NaN                              NaN

          OMENRYP_FULLTIME_POOLED_SUPP OMENRAP_FULLTIME_POOLED_SUPP  \
```

```
2011 0                             NaN                          NaN
     1                             NaN                          NaN
     2                             NaN                          NaN
     3                             NaN                          NaN
     4                             NaN                          NaN
…                                 …                           …
2013 7799                          NaN                          NaN
     7800                          NaN                          NaN
     7801                          NaN                          NaN
     7802                          NaN                          NaN
     7803                          NaN                          NaN

          OMAWDP8_FULLTIME_POOLED_SUPP OMENRUP_FULLTIME_POOLED_SUPP  \
2011 0                             NaN                          NaN
     1                             NaN                          NaN
     2                             NaN                          NaN
     3                             NaN                          NaN
     4                             NaN                          NaN
…                                 …                           …
2013 7799                          NaN                          NaN
     7800                          NaN                          NaN
     7801                          NaN                          NaN
     7802                          NaN                          NaN
     7803                          NaN                          NaN

          OMENRYP_PARTTIME_POOLED_SUPP OMENRAP_PARTTIME_POOLED_SUPP  \
2011 0                             NaN                          NaN
     1                             NaN                          NaN
     2                             NaN                          NaN
     3                             NaN                          NaN
     4                             NaN                          NaN
…                                 …                           …
2013 7799                          NaN                          NaN
     7800                          NaN                          NaN
     7801                          NaN                          NaN
     7802                          NaN                          NaN
     7803                          NaN                          NaN

          OMAWDP8_PARTTIME_POOLED_SUPP OMENRUP_PARTTIME_POOLED_SUPP
2011 0                             NaN                          NaN
     1                             NaN                          NaN
     2                             NaN                          NaN
     3                             NaN                          NaN
     4                             NaN                          NaN
…                                 …                           …
2013 7799                          NaN                          NaN
     7800                          NaN                          NaN
```

```
7801                      NaN                      NaN
7802                      NaN                      NaN
7803                      NaN                      NaN

[30832 rows x 1977 columns]
```

[20]:
```
# Now we have the indices as the year so we know what observations are from␣
↪what year. You should know that
# concatenation also has inner and outer method. If you are concatenating two␣
↪dataframes that do not have
# identical columns, and choose the outer method, some cells will be NaN. If␣
↪you choose to do inner, then some
# observations will be dropped due to NaN values. You can think of this as␣
↪analogous to the left and right
# joins of the merge() function.
```

Now you know how to merge and concatenate datasets together. You will find such functions very useful for combining data to get more complex or complicated results and to do analysis with. A solid understanding of how to merge data is absolutely essentially when you are procuring, cleaning, and manipulating data. It's worth knowing how to join different datasets quickly, and the different options you can use when joining datasets, and I would encourage you to check out the pandas docs for joining and concatenating data.