03-01: Merging DataFrames

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.

Venn Diagram

Venn Diagrams

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.

Venn Diagrams: Full Outer Join (or 'Union')

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.

Union

Venn Diagrams: Full Inner Join / Left Inner Join / Right Inner Join (or 'Intersection')

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:
Intersection

Venn Diagrams: Left Outer Join & Right Outer Join

TO DO ! FIND IMAGES FOR THIS!

Coding in Pandas

Coding in Pandas: Merging on a particular index using `set_index()` and `pd.merge()`

In [1]:

```
# 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. (List of Dicts)
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
```

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** .

Coding in Pandas: Full Outer Join to determine all the students & staff in a university.

In [2]:

```
# 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 t
o use the left and right
# indices as the joining columns.

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

Out[2]:

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

Coding in Pandas: Inner Join to determine those who are both student and staff.

In [3]:

```
# 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 AN
D a staff, we could set the
# how attribute to inner. Again, we set both left and right indices to be true a
s the joining columns
pd.merge(staff_df, student_df, how='inner', left_index=True, right_index=True)
```

Out[3]:

	Role	School
Name		
Sally	Course liasion	Engineering
James	Grader	Business

Coding in Pandas: Using a 'Left Outer Join' to find a list of all staff (and their schools IF they were also students)

In [4]:

```
# 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)
```

Out[4]:

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

Coding in Pandas: Using a 'Right Outer Join' to find a list of all students (and their roles IF they were also staff).

In [5]:

```
# 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)
```

Out[5]:

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

Coding in Pandas: Using `pd.merge(... on = "index"...)` parameter.

In [6]:

```
# 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')
```

Out[6]:

	Name	Role	School
0	Sally	Course liasion	Engineering
1	James	Grader	Business
2	Mike	NaN	Law

In [ ]:

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

Conflicts between DataFrames

When 2 dataframes with same column names have different information. In the below example, James is both at "Washington Avenue" and "1024 Billard Avenue".

In [7]:

```
# So what happens when we have conflicts between the DataFrames? Let's take a lo
ok 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 Av
enue, and these locations just
# happen to be right outside my window as I film this. But for the student DataF
rame, 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 info
rmation, and the _y is always
# the right DataFrame information.

# Here, if we want all the staff information regardless of whether they were stu
dents 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')
```

Out[7]:

	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

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.

Multi-Indexing & Multiple Columns (`pd.merge(... on = ['list of cols']...)`)

In [8]:

```
# Before we leave merging of DataFrames, let's talk about multi-indexing and mul
tiple columns. It's quite
# possible that the first name for students and staff might overlap, but the las
t name might not. In this
# case, we use a list of the multiple columns that should be used to join keys f
rom both dataframes on the on
# parameter. Recall that the column name(s) assigned to the on parameter needs t
o 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'])
```

Out[8]:

	First Name	Last Name	Role	School
0	Sally	Brooks	Course liasion	Engineering

In [ ]:

```
# Joining dataframes through merging is incredibly common, and you'll need to kn
ow how to pull data from
# different sources, clean it, and join it for analysis. This is a staple not on
ly of pandas, but of database
# technologies as well.
```

Concatenating DataFrames using `pd.concat()`

Imagine you have a dataset that tracks some information over the years. And **each year's record is a separate CSV and every CSV for 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.

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.

SideTrack: `%%capture`

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

In [ ]:

```
# 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
```

In [10]:

```
%%capture
# Suppress output while loading CSV files because there are errors in them.
#THAT HAS to be at the very beginning of the cell.
df_2011 = pd.read_csv("datasets/college_scorecard/MERGED2011_12_PP.csv", error_bad_lines=False)
# Tell pandas to not throw an error.
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)
```

In [11]:

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

Out[11]:

	UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZIP	ACCREDAGENCY
0	100654.0	100200.0	1002	Alabama A & M University	Normal	AL	35762	NaN
1	100663.0	105200.0	1052	University of Alabama at Birmingham	Birmingham	AL	35294-0110	NaN
2	100690.0	2503400.0	25034	Amridge University	Montgomery	AL	36117-3553	NaN

3 rows × 1977 columns

In [12]:

```
# 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
```

In [13]:

```
# That's a bit surprising that the number of schools in the scorecard for 2011 i
s almost double that of the
# next two years. But let's not worry about that. Instead, let's just put all th
ree dataframes in a list and
# call that list frames and pass the list into the concat() function Let's see w
hat it looks like

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

Out[13]:

	UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZIP	ACC
0	100654.0	100200.0	1002	Alabama A & M University	Normal	AL	35762	
1	100663.0	105200.0	1052	University of Alabama at Birmingham	Birmingham	AL	35294-0110	
2	100690.0	2503400.0	25034	Amridge University	Montgomery	AL	36117-3553	
3	100706.0	105500.0	1055	University of Alabama in Huntsville	Huntsville	AL	35899	
4	100724.0	100500.0	1005	Alabama State University	Montgomery	AL	36104-0271	
5	100751.0	105100.0	1051	The University of Alabama	Tuscaloosa	AL	35487-0166	
6	100760.0	100700.0	1007	Central Alabama Community College	Alexander City	AL	35010	
7	100812.0	100800.0	1008	Athens State University	Athens	AL	35611	
8	100830.0	831000.0	8310	Auburn University at Montgomery	Montgomery	AL	36117-3596	
9	100858.0	100900.0	1009	Auburn University	Auburn	AL	36849	
10	100937.0	101200.0	1012	Birmingham Southern College	Birmingham	AL	35254	
11	101028.0	1218200.0	12182	Chattahoochee Valley Community College	Phenix City	AL	36869	
12	101073.0	1055400.0	10554	Concordia College Alabama	Selma	AL	36701	
13	101116.0	1303906.0	13039	South University-Montgomery	Montgomery	AL	36116	
14	101143.0	101500.0	1015	Enterprise State Community College	Enterprise	AL	36330-1300	
15	101161.0	106000.0	1060	Coastal Alabama Community College	Bay Minette	AL	36507-2698	
16	101189.0	100300.0	1003	Faulkner University	Montgomery	AL	36109-3390	
17	101240.0	101700.0	1017	Gadsden State Community College	Gadsden	AL	35903	

	UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZIP	ACC
18	101277.0	4187200.0	41872	New Beginning College of Cosmetology	Albertville	AL	35951	
19	101286.0	101800.0	1018	George C Wallace Community College-Dothan	Dothan	AL	36303-9234	
20	101295.0	787100.0	7871	George C Wallace State Community College-Hance...	Hanceville	AL	35077-2000	
21	101301.0	569900.0	5699	George C Wallace State Community College-Selma	Selma	AL	36703-2808	
22	101365.0	962107.0	9621	Herzing University-Birmingham	Birmingham	AL	35209	
23	101435.0	101900.0	1019	Huntingdon College	Montgomery	AL	36106-2148	
24	101453.0	2199700.0	21997	Heritage Christian University	Florence	AL	35630-9977	
25	101462.0	526000.0	5260	J. F. Drake State Community and Technical College	Huntsville	AL	35811	
26	101480.0	102000.0	1020	Jacksonville State University	Jacksonville	AL	36265	
27	101499.0	102100.0	1021	Jefferson Davis Community College	Brewton	AL	36426	
28	101505.0	102200.0	1022	Jefferson State Community College	Birmingham	AL	35215-3098	
29	101514.0	101300.0	1013	John C Calhoun State Community College	Tanner	AL	35671	
...	...	...	...	...	...	...	...	
7774	45897302.0	145992.0	1459	Strayer University-Katy	Houston	TX	77079	
7775	45897303.0	145994.0	1459	Strayer University-Northwest Houston	Houston	TX	77064	
7776	45897304.0	145995.0	1459	Strayer University-Plano	Plano	TX	75093	
7777	45897305.0	10145900.0	1459	Strayer University-Cedar Hill	Cedar Hill	TX	75104	

	UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZIP	ACC
7778	45897306.0	10145901.0	1459	Strayer University-North Dallas	Dallas	TX	75251	
7779	45897307.0	10145908.0	1459	Strayer University-San Antonio	San Antonio	TX	78216	
7780	45897308.0	10145909.0	1459	Strayer University-Stafford	Stafford	TX	77477	
7781	45920401.0	4169701.0	41697	Unitek College	Fremont	CA	945383185	
7782	46163401.0	4176201.0	41762	Twin Rivers Adult School - Grand Avenue Center	Sacramento	CA	95838-3654	
7783	46163402.0	4176202.0	41762	Twin Rivers Adult School - Arnold Avenue Center	McClellan	CA	956521025	
7784	46163403.0	4176205.0	41762	Greater Sacramento Urban League	Sacramento	CA	958383738	
7785	46163404.0	4176206.0	41762	I-TAP	Sacramento	CA	95841-2989	
7786	47647001.0	4088301.0	40883	WestMed College - Merced	Merced	CA	953400000	
7787	47657701.0	4184801.0	41848	Vantage College	El Paso	TX	799064951	
7788	47657702.0	4184802.0	41848	Vantage College	Austin	TX	787523733	
7789	47691101.0	4205801.0	42058	SAE Institute of Technology San Francisco	Emeryville	CA	94608	
7790	47701101.0	10145905.0	1459	Strayer University-Bloomington Campus	Bloomington	MN	554311411	
7791	47702001.0	10145903.0	1459	Strayer University-Schaumburg Campus	Schaumburg	IL	601735081	
7792	47702002.0	10145902.0	1459	Strayer University-Downers Grove Campus	Downers Grove	IL	605151169	
7793	47702003.0	10145906.0	1459	Strayer University-Aurora Campus	Aurora	IL	605066220	
7794	48065701.0	869423.0	8694	Rasmussen College - Overland Park	Overland Park	KS	662102786	
7795	48154401.0	4220901.0	42209	National Personal Training Institute of Cleveland	Highland Heights	OH	44143	

	UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZIP	ACC
7796	48158001.0	10145911.0	1459	Strayer University-St Louis-Brentwood	Brentwood	MO	63144	
7797	48285701.0	157102.0	1571	Georgia Military College-Augusta Campus	Martinez	GA	30907	
7798	48285702.0	157103.0	1571	Georgia Military College-Fairburn Campus	Fairburn	GA	30213	
7799	48285703.0	157107.0	1571	Georgia Military College-Columbus Campus	Columbus	GA	31909	
7800	48285704.0	157101.0	1571	Georgia Military College-Valdosta Campus	Valdosta	GA	31605	
7801	48285705.0	157105.0	1571	Georgia Military College-Warner Robins Campus	Warner Robins	GA	31093	
7802	48285706.0	157100.0	1571	Georgia Military College-Online	Milledgeville	GA	31061	
7803	48285707.0	157103.0	1571	Georgia Military College-Stone Mountain	Stone Mountain	GA	30083	

30832 rows × 1977 columns

In [14]:

```
# 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)
```

Out[14]:

```
30832
```

In [15]:

```
# 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 anym
ore! 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 dataframe
s into the keys parameter

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

Out[15]:

		UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZIP
2011	**0**	100654.0	100200.0	1002	Alabama A & M University	Normal	AL	35762
	1	100663.0	105200.0	1052	University of Alabama at Birmingham	Birmingham	AL	35294 0110
	2	100690.0	2503400.0	25034	Amridge University	Montgomery	AL	36117 3553
	3	100706.0	105500.0	1055	University of Alabama in Huntsville	Huntsville	AL	35899
	4	100724.0	100500.0	1005	Alabama State University	Montgomery	AL	36104 0271
	5	100751.0	105100.0	1051	The University of Alabama	Tuscaloosa	AL	35487 0166
	6	100760.0	100700.0	1007	Central Alabama Community College	Alexander City	AL	35010
	7	100812.0	100800.0	1008	Athens State University	Athens	AL	35611
	8	100830.0	831000.0	8310	Auburn University at Montgomery	Montgomery	AL	36117 3596
	9	100858.0	100900.0	1009	Auburn University	Auburn	AL	36849
	10	100937.0	101200.0	1012	Birmingham Southern College	Birmingham	AL	35254
	11	101028.0	1218200.0	12182	Chattahoochee Valley Community College	Phenix City	AL	36869
	12	101073.0	1055400.0	10554	Concordia College Alabama	Selma	AL	36701
	13	101116.0	1303906.0	13039	South University-Montgomery	Montgomery	AL	36116
	14	101143.0	101500.0	1015	Enterprise State Community College	Enterprise	AL	36330 1300
	15	101161.0	106000.0	1060	Coastal Alabama Community College	Bay Minette	AL	36507 2698
	16	101189.0	100300.0	1003	Faulkner University	Montgomery	AL	36109 3390
	17	101240.0	101700.0	1017	Gadsden State Community College	Gadsden	AL	35903

		UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZI
	18	101277.0	4187200.0	41872	New Beginning College of Cosmetology	Albertville	AL	3595
	19	101286.0	101800.0	1018	George C Wallace Community College-Dothan	Dothan	AL	36303 923
	20	101295.0	787100.0	7871	George C Wallace State Community College-Hance...	Hanceville	AL	35077 200
	21	101301.0	569900.0	5699	George C Wallace State Community College-Selma	Selma	AL	36703 280
	22	101365.0	962107.0	9621	Herzing University-Birmingham	Birmingham	AL	3520
	23	101435.0	101900.0	1019	Huntingdon College	Montgomery	AL	36106 214
	24	101453.0	2199700.0	21997	Heritage Christian University	Florence	AL	35630 997
	25	101462.0	526000.0	5260	J. F. Drake State Community and Technical College	Huntsville	AL	3581
	26	101480.0	102000.0	1020	Jacksonville State University	Jacksonville	AL	3626
	27	101499.0	102100.0	1021	Jefferson Davis Community College	Brewton	AL	3642
	28	101505.0	102200.0	1022	Jefferson State Community College	Birmingham	AL	35215 309
	29	101514.0	101300.0	1013	John C Calhoun State Community College	Tanner	AL	3567
...	**...**	...	...	...	...	...	...	..
2013	**7774**	45897302.0	145992.0	1459	Strayer University-Katy	Houston	TX	7707
	7775	45897303.0	145994.0	1459	Strayer University-Northwest Houston	Houston	TX	7706
	7776	45897304.0	145995.0	1459	Strayer University-Plano	Plano	TX	7509
	7777	45897305.0	10145900.0	1459	Strayer University-Cedar Hill	Cedar Hill	TX	7510

	UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZIP
7778	45897306.0	10145901.0	1459	Strayer University-North Dallas	Dallas	TX	7525
7779	45897307.0	10145908.0	1459	Strayer University-San Antonio	San Antonio	TX	7821
7780	45897308.0	10145909.0	1459	Strayer University-Stafford	Stafford	TX	7747
7781	45920401.0	4169701.0	41697	Unitek College	Fremont	CA	94538318
7782	46163401.0	4176201.0	41762	Twin Rivers Adult School - Grand Avenue Center	Sacramento	CA	95838 365
7783	46163402.0	4176202.0	41762	Twin Rivers Adult School - Arnold Avenue Center	McClellan	CA	95652102
7784	46163403.0	4176205.0	41762	Greater Sacramento Urban League	Sacramento	CA	95838373
7785	46163404.0	4176206.0	41762	I-TAP	Sacramento	CA	95841 298
7786	47647001.0	4088301.0	40883	WestMed College - Merced	Merced	CA	95340000
7787	47657701.0	4184801.0	41848	Vantage College	El Paso	TX	79906495
7788	47657702.0	4184802.0	41848	Vantage College	Austin	TX	78752373
7789	47691101.0	4205801.0	42058	SAE Institute of Technology San Francisco	Emeryville	CA	9460
7790	47701101.0	10145905.0	1459	Strayer University-Bloomington Campus	Bloomington	MN	55431141
7791	47702001.0	10145903.0	1459	Strayer University-Schaumburg Campus	Schaumburg	IL	60173508
7792	47702002.0	10145902.0	1459	Strayer University-Downers Grove Campus	Downers Grove	IL	60515116
7793	47702003.0	10145906.0	1459	Strayer University-Aurora Campus	Aurora	IL	60506622
7794	48065701.0	869423.0	8694	Rasmussen College - Overland Park	Overland Park	KS	66210278
7795	48154401.0	4220901.0	42209	National Personal Training Institute of Cleveland	Highland Heights	OH	4414

	UNITID	OPEID	OPEID6	INSTNM	CITY	STABBR	ZIP
7796	48158001.0	10145911.0	1459	Strayer University-St Louis-Brentwood	Brentwood	MO	63144
7797	48285701.0	157102.0	1571	Georgia Military College-Augusta Campus	Martinez	GA	30907
7798	48285702.0	157103.0	1571	Georgia Military College-Fairburn Campus	Fairburn	GA	30213
7799	48285703.0	157107.0	1571	Georgia Military College-Columbus Campus	Columbus	GA	31909
7800	48285704.0	157101.0	1571	Georgia Military College-Valdosta Campus	Valdosta	GA	31605
7801	48285705.0	157105.0	1571	Georgia Military College-Warner Robins Campus	Warner Robins	GA	31093
7802	48285706.0	157100.0	1571	Georgia Military College-Online	Milledgeville	GA	31061
7803	48285707.0	157103.0	1571	Georgia Military College-Stone Mountain	Stone Mountain	GA	30083

30832 rows × 1977 columns

In [ ]:

```
# 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.