<u>Unit 3</u>: Introduction to the Dataframe Manipulation and Data Cleaning

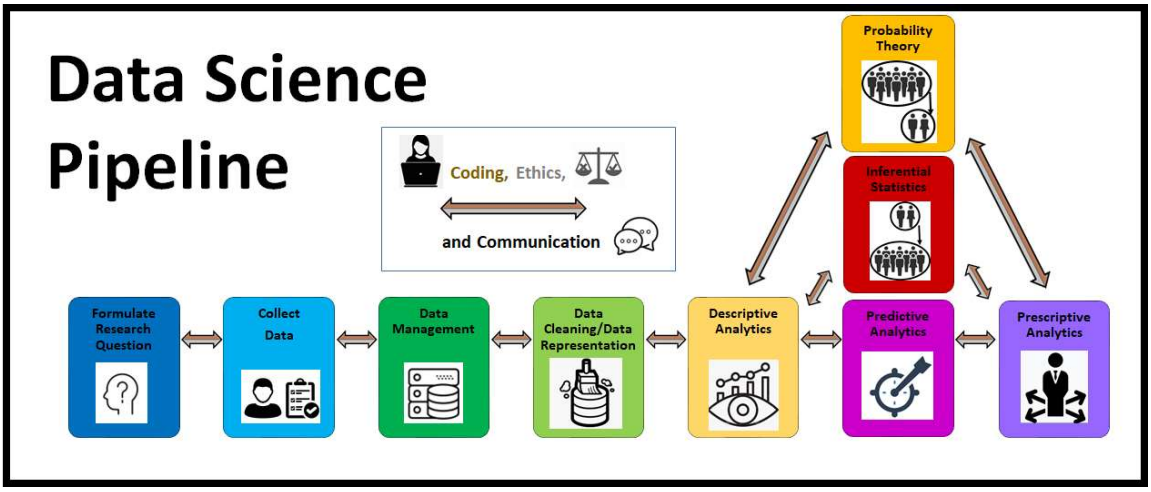
<u>Case Study</u>: Artificial UIUC Course Catalog Dataset Exploration

- *We will build an 'artificial' **dataframe** of UIUC class information. We will explore different way to **represent**, **manipulate**, and **'clean'** this dataframe.*

<u>Case Study</u>: Melanoma Mortality Rate Dataset

- *Use the skills that we have learned so far to answer: is there an <u>association</u> between <u>states in the U.S.</u> and <u>melanoma mortality rate</u>?*

<u>Purpose of this Lecture:</u>



In this lecture we will cover the following topics.

<u>Case Study</u>: Artificial UIUC Course Catalog Dataset Exploration

1. Common types of **Python objects.**
2. How to **<u>create</u> a dataframe** "from scratch"?
3. How to **<u>select</u> a single column** from a dataframe.
4. How to **<u>add</u> a new column** to a dataframe?
5. How to **<u>write</u> a dataframe** to a csv file?
6. How to create a **<u>subset</u> of rows and/or columns of a dataframe?**
7. How to **<u>filter rows</u> of a dataframe** on a **column entry conditions?**
8. How to **<u>summarize</u> (or aggregate) columns of dataframes**?
9. How to **<u>concatenate</u> (ie. "stack") two dataframes** on top of eachother?
10. How to **<u>merge</u> (ie. "join") two dataframes** based on common entries in a column from both dataframes?
11. How to **<u>sort</u> a dataframes** by a specified column?
12. How can we **<u>overwrite a single entry</u>** in a dataframe?

13. What is a **missing value (ie. NaN)** in Python?
14. How do we **find missing values** in a dataframe?
15. How do we **drop all rows with missing values** from a dataframe?

Case Study: Is there an association between states in the U.S. and melanoma mortality rates?

16. Use the skills that we have learned so far in this class to answer this question.

Additional resources:

- Chapter 3 in J. VanderPlas (2016) *Python Data Science Handbook*, https://jakevdp.github.io/PythonDataScienceHandbook/

Unit 3: Introduction to Dataframe Manipulation and Data Cleaning

Review

We have seen that pandas data frames have a spreadsheet like structure with the following characteristics:

- **columns** correspond to different variables and have a single type, either numerical (integer, floating point, complex) or categorical (character strings or boolean).
- **rows** are labeled by an **index** that identifies individual elements, which may be subjects, different time points, subject visits at different time points, products, or any other basic unit under study.

This basic spreadsheet structure is made abundently clear by how we can use the pandas .read_csv function to read an Excel-style comma separated file directly into a pandas data frame.

We have also seen that there are other functions that operate on data frames either to extract their attributes (e.g. the pandas .head() function) or perform other operations like summing, averaging or graphing.

In this section we delve further into the data frame structure, investigating:

- How to build up data frames from simpler objects;
- How to import and export data files;
- How to extract subsets of the data and refer to individual elements in a data frame;
- How to add new data;
- How to combine data from multiple sources;
- How to sort data by specific variables in the data frame.
- How missing data are represented, and how we can process them.

Preliminaries: Importing pandas functions for dataframe manipulation.

The functions contained in the **pandas package** are the primary way that we go about **manipulating dataframes** (remember that a dataframe is a pandas object) as well as **cleaning dataframes**.

Let's import ALL of the functions from pandas below.

In [1]:
```
import pandas as pd
```

Case Study: Artificial UIUC Course Catalog Dataset Exploration

---

1. Common types of Python objects.

1a. Creating Lists

Let's first create two **list** objects below.

In [2]:
```
courses = ['cs105', 'stat107', 'stat207', 'adv307', 'hist407']
courses
```

Out[2]:
```
['cs105', 'stat107', 'stat207', 'adv307', 'hist407']
```

In [3]:
```
type(courses)
```

Out[3]:
```
list
```

In [4]:
```
enrollment = [345, 197, 53, 38, 26]
enrollment
```

Out[4]:
```
[345, 197, 53, 38, 26]
```

In [5]:
```
type(enrollment)
```

Out[5]:
```
list
```

By using the **type()** function we verified that these are both lists. Let's check what kind of objects are contained in these two lists.

A **list** can contain any combination of objects and is always enclosed in **brackets** when we create one. Remember to use commas to separate the entries in the list.

1b. Subsets of Lists

Let's individually extract and print the first three entries in the courses list below. Notice that in Python:

- the first entry is represented with an index of 0,
- the second entry is represented with an index of 1,
- the third entry is represented with an index of 2, ...

In [6]:
```
courses[0]
```

Out[6]:
```
'cs105'
```

In [7]:
```
courses[1]
```

Out[7]:
```
'stat107'
```

In [8]:
```
courses[2]
```

Out[8]:
```
'stat207'
```

1c. Strings

We can see that the entries in the courses enrollments are listed as **string objects**.

In [9]:
```
type(courses[0])
```

Out[9]:
```
str
```

1d. Integers

We can see that the entries in the enrollment list are **integer objects.**

In [10]:
```
type(enrollment[4])
```

Out[10]:
```
int
```

1e. Creating Dictionaries

Now let's create a **dictionary object** below.

In [11]:
```
course_dictionary={'course': courses, 'enrolled': enrollment}
course_dictionary
```

Out[11]:
```
{'course': ['cs105', 'stat107', 'stat207', 'adv307', 'hist407'],
 'enrolled': [345, 197, 53, 38, 26]}
```

In [12]:
```
type(course_dictionary)
```

Out[12]:
```
dict
```

A dictionary is a set of **ordered pairs** of **keys** and **values**. For instance,

- the 'course' key in course_dictionary corresponds to the courses value and
- the 'enrolled' key in course_dictionary corresponds to the enrollment value.

The structure for creating a dictionary is:

{'key1': value1, 'key2': value2, 'key3: value3, ...}

1f More about Python functions

Pay close attention to the different types of brackets.

- '( )' enclose function arguments
- '[ ]' enclose elements in a *list* or *array*
- '{ }' enclose elements in a *dictionary* {'key1': value1, 'key2': value2, ...}

2. Create a new dataframe "from scratch".

Let's use these objects we just created to create a new dataframe "from scratch" (ie. rather than reading one in from a data file like a csv).

We can use the pandas **Dataframe()** function to create a dataframe from a dictionary.

Notice how each of the **keys become a column name** and each of the **values become a column of data**.

In [13]:
```
#Both of these lines of code below do the same thing.
littledf = pd.DataFrame(course_dictionary)
littledf = pd.DataFrame({'course': courses, 'enrolled': enrollment})
littledf
```

Out[13]:

	course	enrolled
0	cs105	345
1	stat107	197
2	stat207	53
3	adv307	38
4	hist407	26

3. Select a single column from a dataframe.

Remember we can **select an already existing column** from a dataframe by using brackets and the name of the column we want.

dataframe_name['column_name']

```
In [14]: littledf['enrolled']
```

```
Out[14]: 0    345
         1    197
         2     53
         3     38
         4     26
         Name: enrolled, dtype: int64
```

4. Add a column to a dataframe.

We can **create an new column from a dataframe** by using brackets and the name of the column we want to create on the left, and then an object with the new data that we want to have in this new column on the right.

dataframe_name['column_name']= new data

```
In [15]: littledf['college'] = ['ENGR', 'LAS', 'LAS', 'MEDIA', 'LAS']
         littledf
```

Out[15]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS

5. How to write a dataframe to a csv file.

The reverse operation is to write an internal data frame to an external file, perhaps after some data processing to merge data from multiple sources. Here we export the 'littledf' data to an external csv file using the pandas.DataFrame.to_csv function.

```
In [16]: littledf.to_csv('courses.csv')
```

Go the folder that this Jupyter notebook is saved in, and you will see that a **new csv** file was created called 'courses.csv' containing the data that was in littledf.

6. How to create a subset of rows and/or columns of a dataframe?

(Given that you know the indices of the rows and columns that you're looking for.)

Using the **.iloc** (index location) attribute, we can refer to specific elements or "slices" of elements in the data frame.

Here, again, is our sample data frame in full:

In [17]:
```
littledf
```

Out[17]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS

6a. Selecting a single entry (using the indices).

In this 5 x 3 array the rows are numbered 0, 1, ..., 4 and the columns are numbered 0,1,2. We can extract the upper left element using .iloc:

In [18]:
```
littledf.iloc[0,0]
```

Out[18]:
```
'cs105'
```

We extract the element in row 3, column 2 as:

In [19]:
```
littledf.iloc[3,2]
```

Out[19]:
```
'MEDIA'
```

Another way is to extract the column (using the name) first, and then select the index of the entry in the column you want.

In [20]:
```
littledf['college'][3]
```

Out[20]:
```
'MEDIA'
```

In [21]:
```
littledf['college']
```

Out[21]:
```
0     ENGR
1      LAS
2      LAS
3    MEDIA
4      LAS
Name: college, dtype: object
```

6b. Selecting a subset of dataframe entries (using a *range* of row indices and/or columns).

We can extract a slice of more than one element using the sequence notation i:j:k to refer to indices running from i to j-k using step-size k. If we leave out the step it is assumed k=1. If we leave out the range elements the sequence covers the whole range.

Here's an example where we can extract the middle three rows of the data frame. Note that "1:4" results in the inclusion of rows 1, 2 and 3 but not 4!

In [22]:
```
littledf.iloc[1:4,:]
```

Out[22]:

	course	enrolled	college
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA

If we wanted to include rows 0-3 we can use the sequence ":4", which includes all rows before the row with index=4.

In [23]:
```
littledf.iloc[:4,:]
```

Out[23]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA

If, on the other hand, we wished to include all rows after rows 0 and 1 the sequence "2:" will do this.

In [24]:
```
littledf.iloc[2:,:]
```

Out[24]:

	course	enrolled	college
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS

6c. Selecting a subset of dataframe entries (using a list of row and/or column entries that we want).

We can also specify lists of row and/or column indicies that we want to select.

In [25]:
```
littledf.iloc[[1,2,4],:]
```

Out[25]:

	course	enrolled	college
1	stat107	197	LAS
2	stat207	53	LAS
4	hist407	26	LAS

In [26]:
```
littledf
```

Out[26]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS

In [27]:
```
littledf.iloc[:,[0,2]]
```

Out[27]:

	course	college
0	cs105	ENGR
1	stat107	LAS
2	stat207	LAS
3	adv307	MEDIA
4	hist407	LAS

In [28]:
```
littledf.iloc[[0,3],[2]]
```

Out[28]:

	college
0	ENGR
3	MEDIA

7. How to filter rows of a dataframe based on *column entry conditions*?

In the first line of code, below (6c), we selected only the rows in the dataframe that corresponded to LAS courses. How can select just these rows without tediously having to look up the row indices that correspond to the rows that we want (ie. row 0, 2, and 4).?

In [29]:
```
#Remember what this is
littledf
```

Out[29]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS

In [30]:
```
#Remember what this does
littledf['college']
```

Out[30]:
```
0      ENGR
1       LAS
2       LAS
3     MEDIA
4       LAS
Name: college, dtype: object
```

7a. Creating a series that 'checks' whether a *given condition* is true.

First, observe how we can check each course for whether or not it is an LAS course with an array operation:

In [31]:
```
#This is a condition that we check on the entries of the college column in lit
tle df. Is the entry = 'LAS'?
littledf['college']=='LAS'
```

Out[31]:
```
0    False
1     True
2     True
3    False
4     True
Name: college, dtype: bool
```

In [32]:
```
type(littledf['college']=='LAS')
```

Out[32]:
```
pandas.core.series.Series
```

7b. Filtering rows in a dataframe based on a column condition.

dataframe_name[column entry condition]

The data frame can take this boolean series. as a condition for selecting rows:

In [33]:
```
littledf[littledf['college']=='LAS']
```

Out[33]:

	course	enrolled	college
1	stat107	197	LAS
2	stat207	53	LAS
4	hist407	26	LAS

7c. Combining row filtering (based on a condition) AND THEN selecting a column from the filtered dataframe.

What if we only want the enrollments of the LAS courses?

One way.... do these steps one at a time...

In [34]:
```
smallerdf=littledf[littledf['college']=='LAS']
smallerdf
```

Out[34]:

	course	enrolled	college
1	stat107	197	LAS
2	stat207	53	LAS
4	hist407	26	LAS

In [35]:
```
smallerdf['enrolled']
```

Out[35]:
```
1    197
2     53
4     26
Name: enrolled, dtype: int64
```

Or... do it all at once...

In [36]:
```
littledf[littledf['college']=='LAS']['enrolled']
```

Out[36]:
```
1    197
2     53
4     26
Name: enrolled, dtype: int64
```

Why does this work? Extracting the three row data frame for LAS courses only gives us a shorter three-column data frame. We can refer to the 'enrolled' column of this short data frame in the same way as for the taller original.

7d. Selecting a column from the original dataframe AND THEN row filtering (based on a condition).

By similar logic, we could have gotten to the same result by a different path as follows.

```
In [37]: enrolled_column=littledf['enrolled']
         enrolled_column
```

```
Out[37]: 0    345
         1    197
         2     53
         3     38
         4     26
         Name: enrolled, dtype: int64
```

```
In [38]: enrolled_column[littledf['college']=='LAS']
```

```
Out[38]: 1    197
         2     53
         4     26
         Name: enrolled, dtype: int64
```

Another way... do it all at once...

```
In [39]: littledf['enrolled'][littledf['college']=='LAS']
```

```
Out[39]: 1    197
         2     53
         4     26
         Name: enrolled, dtype: int64
```

7e. Syntax for setting up conditions.

Notice that when we wanted to test if an entry in the 'college' column was **equal to** a 'LAS', we used '==' rather than '='. Here is the distinction between the two:

- Generally, we use '==' when we are setting up a condition in Python.
- We use '=' when we are defining a variable or parameter in Python.

Here are some other **operators** we would use to set up other types of **conditions.**

- **equal to**: ==
- **greater than or equal to**: >=
- **less than or equal to**: <=
- **greater**: >
- **less than**: <

How about a different type of condition, like extracting all the courses with enrollments of at least 50?

```
In [40]: littledf[littledf['enrolled']>=50]
```

Out[40]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS

Or extracting the courses with enrollments less than 50?

In [41]:
```
littledf[littledf['enrolled']<50]
```

Out[41]:

	course	enrolled	college
3	adv307	38	MEDIA
4	hist407	26	LAS

We can extract the record corresponding to a particular course:

In [42]:
```
littledf[littledf['course']=='adv307']
```

Out[42]:

	course	enrolled	college
3	adv307	38	MEDIA

8. How to summarize (or aggregate) columns of dataframes?

In [43]:
```
littledf
```

Out[43]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS

8a. Aggregating a Column

Suppose, first, we want the total enrollment in all classes in our dataframe. Below is one way to get it, using the **.sum()** function.

In [44]:
```
littledf['enrolled'].sum()
```

Out[44]: 659

There are many other column aggregation functions like the following:

- **.min()**
- **.max()**

(we'll learn more function in a later unit).

In [45]:
```
littledf['enrolled'].max()
```

Out[45]: 345

In [46]:
```
littledf['enrolled'].min()
```

Out[46]: 26

8b. Aggregating a filtered dataframe.

Now suppose we want to find the total enrollment of JUST LAS classes. We can do this all at once in the following order:

1. First create a dataframe that is just filtered for 'LAS' classes.
2. Then extract just the 'enrolled' column from this filtered dataframe.
3. Then take the sum of this column you extracted.

In [47]:
```
print('Total LAS Enrollment')
littledf[littledf['college']=='LAS']['enrolled'].sum()
```

```
Total LAS Enrollment
```

Out[47]:
```
276
```

In [48]:
```
#1. WHAT WE DID, STEP-BY-STEP: Just the filtered dataframe below
littledf[littledf['college']=='LAS']
```

Out[48]:

	course	enrolled	college
1	stat107	197	LAS
2	stat207	53	LAS
4	hist407	26	LAS

In [49]:
```
#2. WHAT WE DID, STEP-BY-STEP: Just the enrolled column from the filtered data
frame shown directly above.
littledf[littledf['college']=='LAS']['enrolled']
```

Out[49]:
```
1    197
2     53
4     26
Name: enrolled, dtype: int64
```

In [50]:
```
#3. WHAT WE DID, STEP-BY-STEP: The sum of the the enrolled column shown direct
ly above.
littledf[littledf['college']=='LAS']['enrolled'].sum()
```

Out[50]:
```
276
```

8c. Using the value that you found from an aggregated dataframe in a condition.

Suppose now we wanted to extract the **row** from littledf that has the maximum enrollment. We can first find this largest enrollment size.

In [51]:
```
print("Maximum Enrollment")
littledf['enrolled'].max()
```

```
Maximum Enrollment
```

Out[51]:
```
345
```

We can use this number directly, in a condition that we filter littledf on....

In [52]:
```
littledf[littledf['enrolled']==345]
```

Out[52]:

	course	enrolled	college
0	cs105	345	ENGR

But, because littledf['enrolled'].max()=345, we can use this 'littledf['enrolled'].max()' value directly in the condition and get the same result. Doing it this way is more efficient from a coding perspective.

In [53]:
```
littledf[littledf['enrolled']==littledf['enrolled'].max()]
```

Out[53]:

	course	enrolled	college
0	cs105	345	ENGR

9. How to concatenate (ie. “stack”) two dataframes on top of eachother?

In [54]:
```
littledf
```

Out[54]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS

Suppose we had more enrollment data to add to the data frame, for additional courses. We can use the pandas **concat()** funtion to combine the original data frame with a new data frame containing the additional records. Here we create a new data frame with the hypothetical new data.

In [55]:
```
#Creating a second dataframe that contains two more courses.
moredf = pd.DataFrame({'course': ['math277', 'is417'],
                       'enrolled': [41, 43],
                       'college': ['LAS', 'IS']})
moredf
```

Out[55]:

	course	enrolled	college
0	math277	41	LAS
1	is417	43	IS

Here are the original data frame and the data we wish to add:

In [56]:
```
display(littledf, moredf)
```

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS

	course	enrolled	college
0	math277	41	LAS
1	is417	43	IS

9a. Vertically stacking two dataframes (reseting the index of the new dataframe).

Next we combine them, and specify to ignore the original index values and create a new index for the combined data.

In [57]:
```
fulldf = pd.concat([littledf, moredf], ignore_index=True)
fulldf
```

Out[57]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS
5	math277	41	LAS
6	is417	43	IS

9b. Vertically stacking two dataframes (NOT reseting the index of the new dataframe).

In [58]:
```
pd.concat([littledf, moredf])
```

Out[58]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS
0	math277	41	LAS
1	is417	43	IS

9c. A quick way to add new rows to a dataframe.

A quick way to add new records is using the **append()** function.

In [59]:
```
fulldf
```

Out[59]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS
5	math277	41	LAS
6	is417	43	IS

In [60]:
```
newdf = pd.DataFrame({'course': ['badm210'],
                      'enrolled': [215],
                      'college': ['BUSN']})
newdf
```

Out[60]:

	course	enrolled	college
0	badm210	215	BUSN

In [61]:
```
updateddf = fulldf.append(newdf, ignore_index=True)
updateddf
```

Out[61]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS
5	math277	41	LAS
6	is417	43	IS
7	badm210	215	BUSN

10. How to merge (ie. “join”) two dataframes

Another common scenario is to have more than one source of data on different variables, and we wish to combine data sets for further analysis. As an example, suppose in the previous course list example we had another source with the credit hours for each class. We'd like to add this information.

In [62]:
```
updateddf
```

Out[62]:

	course	enrolled	college
0	cs105	345	ENGR
1	stat107	197	LAS
2	stat207	53	LAS
3	adv307	38	MEDIA
4	hist407	26	LAS
5	math277	41	LAS
6	is417	43	IS
7	badm210	215	BUSN

In [63]:
```
creditdf = pd.DataFrame({'course': ['adv307', 'cs105', 'stat107', 'stat207',
                                    'hist407', 'math277', 'is417', 'badm210'],
                         'credit': [3.0, 3.0, 4.0, 3.0, 4.0, 5.0, 3.0, 3.0]})
creditdf
```

Out[63]:

	course	credit
0	adv307	3.0
1	cs105	3.0
2	stat107	4.0
3	stat207	3.0
4	hist407	4.0
5	math277	5.0
6	is417	3.0
7	badm210	3.0

In this case, we can do a one-to-one join between the two data frames using the pandas **merge()** function. Notice that the order of the courses does not need to be the same; the records are matched based on the shared course name.

In [64]:
```
fullerdf = pd.merge(updateddf, creditdf)
fullerdf
```

Out[64]:

	course	enrolled	college	credit
0	cs105	345	ENGR	3.0
1	stat107	197	LAS	4.0
2	stat207	53	LAS	3.0
3	adv307	38	MEDIA	3.0
4	hist407	26	LAS	4.0
5	math277	41	LAS	5.0
6	is417	43	IS	3.0
7	badm210	215	BUSN	3.0

Often the two data sources will not be in one-to-one correspondence between their records. Then we might need to perform and "many-to-one" merge.

Example: In one data source we have courses and section enrollments. In the other data source we have courses and credit hours. Let's combine them. First we'll create a data frame with the section information.

In [65]:
```
courses = ['cs105', 'cs105', 'stat107', 'badm210', 'badm210']
sections = ['A', 'B', 'A', 'A', 'B']
enrollments = [345, 201, 197, 215, 197]
sectdf = pd.DataFrame({'course': courses,
                       'section': sections,
                       'enrolled': enrollments})
sectdf
```

Out[65]:

	course	section	enrolled
0	cs105	A	345
1	cs105	B	201
2	stat107	A	197
3	badm210	A	215
4	badm210	B	197

We'd like to merge this with the credit information:

In [66]:
```
creditdf
```

Out[66]:

	course	credit
0	adv307	3.0
1	cs105	3.0
2	stat107	4.0
3	stat207	3.0
4	hist407	4.0
5	math277	5.0
6	is417	3.0
7	badm210	3.0

We can try a "default" merge and see what we get:

In [67]:
```
pd.merge(sectdf, creditdf)
```

Out[67]:

	course	section	enrolled	credit
0	cs105	A	345	3.0
1	cs105	B	201	3.0
2	stat107	A	197	4.0
3	badm210	A	215	3.0
4	badm210	B	197	3.0

Did it work? Yes, in the sense that all course sections in the first data frame have now been assigned credit hours. Any course that appears in both data sources gets matched. The courses missing from one or the other we not included.

In some cases we need to specify which variable to use as the matching **key** using the **on=** option:

In [68]:
```
pd.merge(sectdf, creditdf, on='course')
```

Out[68]:

	course	section	enrolled	credit
0	cs105	A	345	3.0
1	cs105	B	201	3.0
2	stat107	A	197	4.0
3	badm210	A	215	3.0
4	badm210	B	197	3.0

11. How to sort a dataframe by a specified column?

In the examples we've been considering, the course names are in no particular order. What if we want the courses to be in alphanumeric order? pandas has a function for that: **.sort_values**. For the syntax see: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_values.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.sort_values.html)

To select a specific column on which to sort we use the **by=** option as in the following example:

In [69]:
```
creditdf
```

Out[69]:

	course	credit
0	adv307	3.0
1	cs105	3.0
2	stat107	4.0
3	stat207	3.0
4	hist407	4.0
5	math277	5.0
6	is417	3.0
7	badm210	3.0

11a. Sort by by descending order

In [70]:
```
creditdf.sort_values(by='course')
```

Out[70]:

	course	credit
0	adv307	3.0
7	badm210	3.0
1	cs105	3.0
4	hist407	4.0
6	is417	3.0
5	math277	5.0
2	stat107	4.0
3	stat207	3.0

Notice how this did not permanently sort the creditdf dataframe.

In [71]:
```
creditdf
```

Out[71]:

	course	credit
0	adv307	3.0
1	cs105	3.0
2	stat107	4.0
3	stat207	3.0
4	hist407	4.0
5	math277	5.0
6	is417	3.0
7	badm210	3.0

Remarks:

1. We can specify more than one variable for sorting, and we can also select various other options such as "ascending=False" (default is "ascending=True"), where to put NaNs in the ordering ("na_position='last'), and whether to sort in-place (overwriting the original object).

1. This operation did **not** replace the original data with sorted data, it merely displayed the sorted data. If we wanted to save this we assign to a new pandas object, or we can sort "in place" as illustrated below.

11b. How to sort the dataframe (and have it remain sorted).

Here we see the effect of in-place sorting.

One way to do this...

```
In [72]: creditdf.sort_values(by='course', inplace=True)  # sorting in place and replac
         ing original
```

```
In [73]: creditdf   # now the original is in sorted order
```

Out[73]:

	course	credit
0	adv307	3.0
7	badm210	3.0
1	cs105	3.0
4	hist407	4.0
6	is417	3.0
5	math277	5.0
2	stat107	4.0
3	stat207	3.0

Another way to do this... (overwrite the dataframe with the changed dataframe.

```
In [74]: creditdf=creditdf.sort_values(by='course')
         creditdf
```

Out[74]:

	course	credit
0	adv307	3.0
7	badm210	3.0
1	cs105	3.0
4	hist407	4.0
6	is417	3.0
5	math277	5.0
2	stat107	4.0
3	stat207	3.0

11c. Sort by ascending order

As a different application, here we sort class sections by enrollment, from lowest to highest.

In [75]:
```
sectdf.sort_values(by='enrolled', ascending=True)
```

Out[75]:

	course	section	enrolled
2	stat107	A	197
4	badm210	B	197
1	cs105	B	201
3	badm210	A	215
0	cs105	A	345

12. How can we overwrite a single entry in a dataframe?

Let's pretend that we are now unsure about the enrollment of section B of badm210. We could overwrite the enrollment entry below by replacing 197 with the string 'unknown.' However, we see below that by adding a string to a column comprised of numbers, this gives us an error when we try to apply a function (like **.sum()**) that only applies to numbers.

In [76]:
```
tmp = sectdf.copy()  # copy of data frame
tmp
```

Out[76]:

	course	section	enrolled
0	cs105	A	345
1	cs105	B	201
2	stat107	A	197
3	badm210	A	215
4	badm210	B	197

In [77]:
```
tmp['enrolled'][4]  # Access the enrollment for badm210 section B
```

Out[77]: 197

In [78]:
```
tmp.iloc[4,2] # another way to access
```

Out[78]: 197

In [79]:
```
tmp.iloc[4,2] = 'unknown'   # coding this element as something else
tmp
```

Out[79]:

	course	section	enrolled
0	cs105	A	345
1	cs105	B	201
2	stat107	A	197
3	badm210	A	215
4	badm210	B	unknown

In [80]:
```
tmp['enrolled'].sum()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-80-25fb9c9d6193> in <module>
----> 1 tmp['enrolled'].sum()

~\Miniconda3\lib\site-packages\pandas\core\generic.py in stat_func(self, axi
s, skipna, level, numeric_only, min_count, **kwargs)
  11408                 name, axis=axis, level=level, skipna=skipna, min_coun
t=min_count
  11409             )
> 11410         return self._reduce(
  11411             func,
  11412             name=name,

~\Miniconda3\lib\site-packages\pandas\core\series.py in _reduce(self, op, nam
e, axis, skipna, numeric_only, filter_type, **kwds)
   4234                 )
   4235             with np.errstate(all="ignore"):
-> 4236                 return op(delegate, skipna=skipna, **kwds)
   4237
   4238     def _reindex_indexer(self, new_index, indexer, copy):

~\Miniconda3\lib\site-packages\pandas\core\nanops.py in _f(*args, **kwargs)
     69             try:
     70                 with np.errstate(invalid="ignore"):
---> 71                     return f(*args, **kwargs)
     72             except ValueError as e:
     73                 # we want to transform an object array

~\Miniconda3\lib\site-packages\pandas\core\nanops.py in nansum(values, axis,
 skipna, min_count, mask)
    507     elif is_timedelta64_dtype(dtype):
    508         dtype_sum = np.float64
--> 509     the_sum = values.sum(axis, dtype=dtype_sum)
    510     the_sum = _maybe_null_out(the_sum, axis, mask, values.shape, min_
count=min_count)
    511

~\Miniconda3\lib\site-packages\numpy\core\_methods.py in _sum(a, axis, dtype,
out, keepdims, initial, where)
     45 def _sum(a, axis=None, dtype=None, out=None, keepdims=False,
     46          initial=_NoValue, where=True):
---> 47     return umr_sum(a, axis, dtype, out, keepdims, initial, where)
     48
     49 def _prod(a, axis=None, dtype=None, out=None, keepdims=False,

TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

13. What is a NaN object?

So what is a better way to indicate to Python that we do not know the enrollment for section B of badm210 that will not give us errors? Let's overwrite the enrollment for this class using another type of filler for unknown or missing values..

In [81]:
```
tmp = sectdf.copy()  # copy of data frame
tmp
```

Out[81]:

	course	section	enrolled
0	cs105	A	345
1	cs105	B	201
2	stat107	A	197
3	badm210	A	215
4	badm210	B	197

In [82]:
```
tmp.iloc[4,2] = None   # coding this element as NaN
tmp
```

Out[82]:

	course	section	enrolled
0	cs105	A	345.0
1	cs105	B	201.0
2	stat107	A	197.0
3	badm210	A	215.0
4	badm210	B	NaN

In [83]:
```
tmp.iloc[4,2]
```

Out[83]:
```
nan
```

Missing data are very common in real data applications. How can we handle them at a basic level? To illustrate, consider the hypothetical section enrollment data. We'll make one element go missing.

We see that the missing value is encoded as NaN (not a number).

What if we wanted to sort by enrollment? We need to specify whether missing values go first or last on the list.

In [84]:
```
tmp.sort_values(by='enrolled', na_position='first')
```

Out[84]:

	course	section	enrolled
4	badm210	B	NaN
2	stat107	A	197.0
1	cs105	B	201.0
3	badm210	A	215.0
0	cs105	A	345.0

By default, many functions will skip data with missing values. Often this makes sense, but not always!

Now when we try to take the same summation of the enrollment, the NaN value tells us enrolled column to skip this value and just sum the remaining values.

In [85]:
```
tmp['enrolled'].sum()
```

Out[85]:
```
958.0
```

14. How do we find missing values in a dataframe (basic)?

The 'DataFrame.isna' function can scan a data frame for missing values. 'DataFrame.notna' scans for non-missing values.

In [86]:
```
tmp.isna()
```

Out[86]:

	course	section	enrolled
0	False	False	False
1	False	False	False
2	False	False	False
3	False	False	False
4	False	False	True

We can take the sum of each column above. When we sum a column of boolean values, Python automatically translates a True to a 1 and a False to a 0.

Therefore, the **.sum()** function counted 1 True (ie. a missing value) in the enrolled column and 0 Trues (ie. missing values) in the other columns.

In [87]:
```
tmp.isna().sum()
```

Out[87]:
```
course      0
section     0
enrolled    1
dtype: int64
```

15. How do we drop all rows with missing values from a dataframe?

If we want to analyze only the data with complete information the 'DataFrame.dropna' function can extract the complete data for us.

In [88]:
```
tmp.dropna()
```

Out[88]:

	course	section	enrolled
0	cs105	A	345.0
1	cs105	B	201.0
2	stat107	A	197.0
3	badm210	A	215.0

Case Study: Melanoma Mortality Rate Dataset

Use the skills that we have learned so far to answer:

Is there an association between states in the U.S. and melanoma mortality rate?

Read the melanoma csv into a dataframe df

In [89]:
```
df=pd.read_csv('USmelanoma.csv')
```

In [90]:
```
df.head()
```

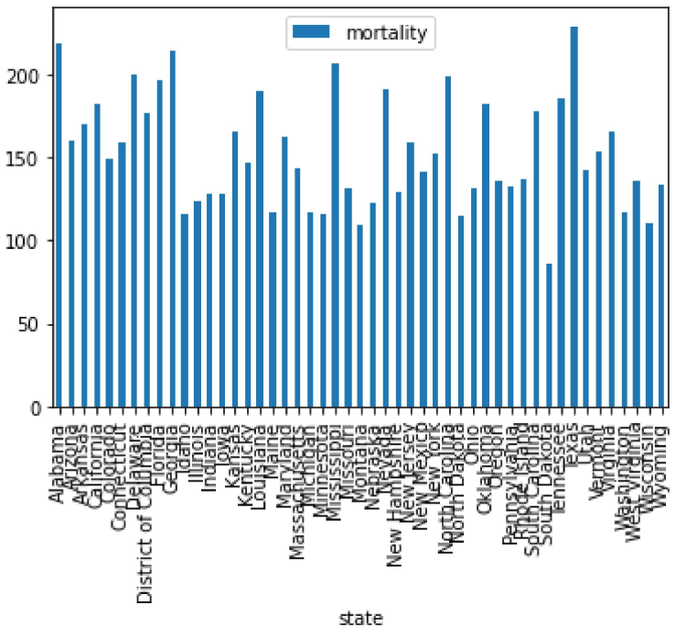
Out[90]:

	state	mortality	latitude	longitude	ocean
0	Alabama	219	33.0	-87.0	1
1	Arizona	160	34.5	-112.0	0
2	Arkansas	170	35.0	-92.5	0
3	California	182	37.5	-119.5	1
4	Colorado	149	39.0	-105.5	0

Let's first plot mortality rates across different states, in alphabetical order.

In [91]:
```
import matplotlib.pyplot as plt
```

In [92]:
```
df.plot.bar(x='state', y='mortality')
plt.show()
```



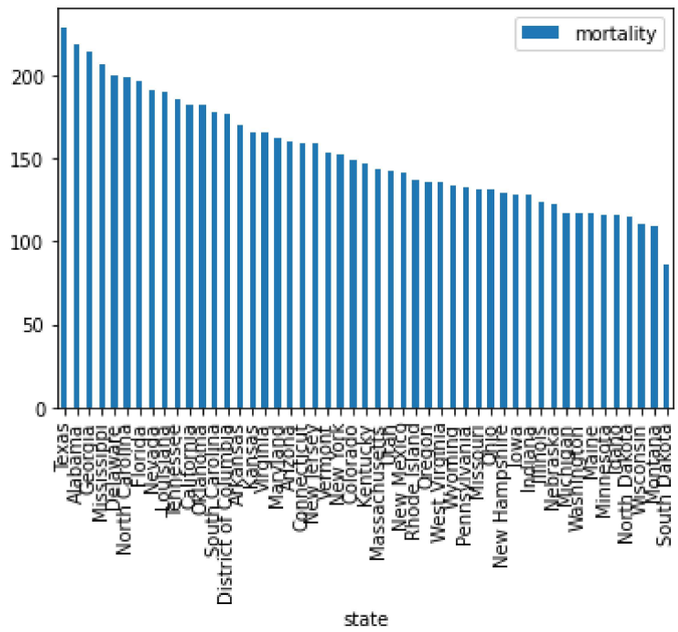
It will be easier to interpret and compare if we sort by mortality rates.

In [93]:
```
dfsorted = df.sort_values(by='mortality', ascending=False)
dfsorted.head()
```

Out[93]:

	state	mortality	latitude	longitude	ocean
41	Texas	229	31.5	-98.0	1
0	Alabama	219	33.0	-87.0	1
9	Georgia	214	33.0	-83.5	1
22	Mississippi	207	32.8	-90.0	1
6	Delaware	200	39.0	-75.5	1

In [94]:
```
dfsorted.plot.bar(x='state', y='mortality')
plt.show()
```



Is there an association between states in the U.S. and melanoma mortality rate?

Further analysis: What additional insights can you describe about the relationship between states and melanoma mortality rate?

If you worked for a US health organization, how might these insights be useful?

In [ ]: