DATA HANDLING

Making Sure Our Data Are Loaded

We will continue to use the surveys dataset that we worked with in the last exercise. Let's reopen it:

```
# first make sure pandas is loaded
import pandas as pd
# read in the survey csv
surveys_df = pd.read_csv("surveys.csv")
```

Indexing & Slicing in Python

We often want to work with subsets of a **DataFrame** object. There are different ways to accomplish this including: using labels (column headings), numeric ranges or specific x,y index locations.

Selecting Data Using Labels (Column Headings)

We use square brackets `[]` to select a subset of an Python object. For example, we can select all of data from a column named `species` from the `surveys_df` DataFrame by name:

```
surveys_df['species']
# this syntax, calling the column as an attribute, gives you the same output
surveys_df.species
```

We can also create an new object that contains the data within the species column as follows:

```
# create an object named surveys_species that only contains the species column
surveys_species = surveys_df['species']
```

We can pass a list of column names too, as an index to select columns in that order. This is useful when we need to reorganize our data.

NOTE: If a column name is not contained in the DataFrame, an exception (error) will be raised.

```
# select the species and plot columns from the DataFrame
surveys_df[['species', 'plot']]
# what happens when you flip the order?
surveys_df[['plot', 'species']]
#what happens if you ask for a column that doesn't exist?
surveys_df['speciess']
```

```

```

Extracting Range based Subsets: Slicing

REMINDER: Python Uses 0-based Indexing

Let's remind ourselves that Python uses 0-based indexing. This means that the first element in an object is located at position 0. This is different from other tools like R and Matlab that index elements within objects starting at 1.

```
# Create a list of numbers:
a = [1,2,3,4,5]
```

Slicing Subsets of Rows in Python

Slicing using the `[]` operator selects a set of rows and/or columns from a Data Frame. To slice out a set of rows, you use the following syntax: `data[start:stop]`. When slicing in pandas the start bound is included in the output. The stop bound is one step BEYOND the row you want to select. So if you want to select rows 0, 1 and 2 your code would look like this:

```
# select rows 0,1,2 (but not 3)
surveys_df[0:3]
```

The stop bound in Python is different from what you might be used to in languages like Matlab and R.

```
# select the first, second and third rows from the surveys variable
surveys_df[0:3]
# select the first 5 rows (rows 0,1,2,3,4)
surveys_df[:5]
# select the last element in the list
surveys_df[-1:]
```

We can also reassign values within subsets of our DataFrame. But before we do that, let's make a copy of our DataFrame so as not to modify our original imported data.

```
# copy the surveys dataframe so we don't modify the original DataFrame
surveys_copy = surveys_df


# set the first three rows of data in the DataFrame to 0
surveys_copy[0:3] = 0
```

Next, try the following code:

```
surveys_copy.head()
surveys_df.head()
```

What is the difference between the two data frames?

Referencing Objects vs Copying Objects in Python

We might have thought that we were creating a fresh copy of the `surveys_df` objects when we used the code `surveys_copy = surveys_df`. However the statement y = x doesn't create a copy of our DataFrame. It creates a new variable y that refers to the **same** object x refers to. This means that there is only one object (the DataFrame), and both x and y refer to it. So when we assign the first 3 columns the value of 0 using the `surveys_copy` DataFrame, the `surveys_df` DataFrame is modified too. To create a fresh copy of the `surveys_df`DataFrame we use the syntax y=x.copy().

```
surveys_copy= surveys_df.copy()
```

Slicing Subsets of Rows and Columns in Python

We can select specific ranges of our data in both the row and column directions using either label or integer-based indexing.

- `loc`: indexing via *labels* or *integers*
- `iloc`: indexing via *integers*

To select a subset of rows AND columns from our DataFrame, we can use the `iloc` method. For example, we can select month, day and year (columns 2, 3 and 4 if we start counting at 1), like this:

```
surveys_df.iloc[0:3, 1:4]
```

which gives **output**

```
   month  day  year
0      7   16  1977
1      7   16  1977
2      7   16  1977
```

Notice that we asked for a slice from 0:3. This yielded 3 rows of data. When you ask for 0:3, you are actually telling python to start at index 0 and select rows 0, 1, 2 **up to but not including 3**.

Let's next explore some other ways to index and select subsets of data:

```
# select all columns for rows of index values 0 and 10
surveys_df.loc[[0, 10], :]
# what does this do?
surveys_df.loc[0, ['species', 'plot', 'wgt']]
```

```
# What happens when you type the code below?
surveys_df.loc[[0, 10, 35549], :]
```

NOTE: Labels must be found in the DataFrame or you will get a `KeyError`. The start bound and the stop bound are **included**. When using `loc`, integers *can* also be used, but they refer to the index label and not the position. Thus when you use `loc`, and select 1:4, you will get a different result than using `iloc` to select rows 1:4.

We can also select a specific data value according to the specific row and column location within the data frame using the `iloc` function: `dat.iloc[row,column]`.

```
surveys_df.iloc[2,6]
```

which gives **output**

```
'F'
```

Remember that Python indexing begins at 0. So, the index location [2, 6] selects the element that is 3 rows down and 7 columns over in the DataFrame.

Subsetting Data Using Criteria

We can also select a subset of our data using criteria. For example, we can select all rows that have a year value of 2002.

```
surveys_df[surveys_df.year == 2002]
```

Which produces the following output:

```
       record_id  month  day  year  plot species  sex  wgt
33320      33321      1   12  2002     1      DM    M   44
33321      33322      1   12  2002     1      DO    M   58
33322      33323      1   12  2002     1      PB    M   45
33323      33324      1   12  2002     1      AB  NaN  NaN
33324      33325      1   12  2002     1      DO    M   29
33325      33326      1   12  2002     2      OT    F   26
33326      33327      1   12  2002     2      OT    M   24


...          ...    ...  ...   ...   ...     ...  ...  ...
35541      35542     12   31  2002    15      PB    F   29
35542      35543     12   31  2002    15      PB    F   34
35543      35544     12   31  2002    15      US  NaN  NaN
35544      35545     12   31  2002    15      AH  NaN  NaN
```

```
35545      35546     12   31  2002    15      AH  NaN  NaN
35546      35547     12   31  2002    10      RM    F   14
35547      35548     12   31  2002     7      DO    M   51
35548      35549     12   31  2002     5     NaN  NaN  NaN

[2229 rows x 8 columns]
```

Or we can select all rows that do not contain the year 2002.

```
surveys_df[surveys_df.year != 2002]
```

We can define sets of criteria too:

```
surveys_df[(surveys_df.year >= 1980) & (surveys_df.year <= 1985)]
```