Slicin'

Slicin' and Dicin', Part One

NOTE: To follow the instruction below, make sure you download the zip file found in the Setting Up Your Lab Environment section.

In this lesson, you'll learn how to intuitively select of a subset of data out of your dataframes or series, an operation known as *indexing*. More than half of the top-voted StackOverflow questions on Pandas revolve around this single topic! This shows you how important and potentially confusing indexing can be. Spend adequate time working through this chapter and playing around with the accompanying dataset.

Learning to properly indexing is like learning the ABC's, so if you have a gap in your indexing knowledge, it'll be hard to complete future tasks. But if you're able to master this, you'll have mastered the building blocks of all future material. To get you accustomed to indexing, first use the .read_csv() method you just learned about to load up the direct_marketing.csv file in your course **/Module2/Datasets/** directory:

```
>>> df = pd.read_csv('Datasets/direct_marketing.csv')
>>> df

      recency  history_segment  history  mens  womens    zip_code  newbie
0          10   2) $100 - $200   142.44     1       0  Surburban       0
1           6   3) $200 - $350   329.08     1       1       Rural       1
2           7   2) $100 - $200   180.65     0       1  Surburban       1
3           9   5) $500 - $750   675.83     1       0       Rural       1
4           2     1) $0 - $100    45.34     1       0       Urban       0
5           6   2) $100 - $200   134.83     0       1  Surburban       0
```

Column Indexing

A dataframe is essentially one or more series which have been 'stitched' together into a new data type. Pandas exposes *many* equivalent methods for slicing out those underlying series. You can slice by location, the way you would normally index into a regular Python list. You can slice by label, the way you would normally index into a Python dictionary. And like NumPy arrays, you can also index by boolean masks:

```
>>> df.recency
>>> df['recency']
>>> df[['recency']]
>>> df.loc[:, 'recency']
>>> df.loc[:, ['recency']]
>>> df.iloc[:, 0]
>>> df.iloc[:, [0]]
>>> df.ix[:, 0]
```

Why does Pandas have so many different data access methods? The answer is because there are slight differences between them. The first difference you'll notice from the list above is that in some of the commands, you specify the *name* of the column or series you want to slice: recency. By using the column name in the code, it's very easy to discern what is being pulled, and you don't have to worry about the order of the columns. Doing this lookup of first matching the column name before slicing the column index is marginally slower than directly accessing the column by index.

Once you're ready to move to a production environment, Pandas documentation recommends you use either .loc[], .iloc[], or .ix[] data access methods, which are more optimized. The .loc[] method selects by column label, .iloc[] selects by column index, and .ix[] can be used whenever you want to use a hybrid approach of either. All code in this course will use either the df.recency or df[['recency', ...]] data-access syntax styles for maximum clarity.

Another difference you'll notice is that some of the methods take in a *list* of parameters, e.g.: df[['recency']], df.loc[:, ['recency']], and df.iloc[:, [0]]. By passing in a list of parameters, you can select more than one column to slice. Please be aware that if you use this syntax, even if you only specify a single column, the data type that you'll get back is a *dataframe* as opposed to a *series*. This will be useful for you to know once you start machine learning, so be sure to take down that note.

```
#
# Produces a series object:
>>> df.recency
>>> df['recency']
>>> df.loc[:, 'recency']
>>> df.iloc[:, 0]
>>> df.ix[:, 0]

0        10
1         6
2         7
3         9
4         2
5         6
Name: recency, dtype: int64

#
# Produces a dataframe object:
>>> df[['recency']]
>>> df.loc[:, ['recency']]
>>> df.iloc[:, [0]]

       recency
0           10
1            6
2            7
3            9
4            2
5            6

[64000 rows x 1 columns]
```

Row Indexing

You can use any of the .loc[], .iloc[], or .ix[] methods to do selection by row, noting that the expected order is [row_indexer, column_indexer]:

```
>>> df[0:2]
>>> df.iloc[0:2, :]

   recency history_segment  history  mens  womens   zip_code  newbie channel
0       10  2) $100 - $200   142.44     1       0  Surburban       0   Phone
1        6  3) $200 - $350   329.08     1       1      Rural       1     Web
```

The last important difference is that .loc[] and .ix[] are *inclusive* of the range of values selected, where .iloc[] is non-inclusive. In that sense, df.loc[0:1, :] would select the first *two* rows, but only the first row would be returned using df.iloc[0:1, :].

English