Week #5 - Python Practice

"Why Would You Want to Do That?"

Sometimes in the data science universe you are asked to do something that just doesn't make sense to you. Sometimes your organizational overlords or their minions need some data from you the use of which isn't at all obvious. But yours is to do. (While always being careful about asking "Why?")

In what follows we're going to use the json format more (you're going to write a json data file) and to do some more data merging. And along the way we're going to change the data type of some variables.

Your mission in this Practice, if you decide to accept it, is as follows.

XYZ's marketing department has a lead on a soon to be available new product they are thinking they might sell. It's a new kind of electronic dog or cat washer that will also scent your pet with a desired fragrance, like pumpkin pie spice, vanilla, cheddar, or BBQ. They are thinking they should test the idea with XYZ customers to see what their level of interest might be. The marketing department wants you to provide them a json data set so they can invite customers to take a short concept testing survey.

The data set is to include all XYZ customers you have data on, and the Experian data on whether each is a dog and/or cat "enthusiast." These Experian variables are ZDOGS and ZCATS, and they are coded Y="Living unit has a known dog enthusiast," and U="Unknown." There may be some missing values on these variables. Your data set is to include the value labels on these variables, e.g. "Unknown," not just the Y and U codes. Include the ACCTNO customer identifier in the data, and also the channel through which each customer first became an XYZ customer, "CHANNEL_ACQUISITION." The codes and their labels for this last variable are: CB=Catalog, IB=Internet, RT=Retail. Include the labels for these codes in the data for the XYZ marketing folks, too.

To get started, crank up your Canopy, and import pandas (as pd), numpy (as np, right?), and SQLAlchemy. You'll want the latter if you saved your results from the last Practice in a SQLite3 DB. From pandas import DataFrame, Series, too.

You're going to need the custZData you worked with in the last session. You saved it somewhere, didn't you? Now's the time to load it back into your Canopy session. How you do this depends on how you saved it, of course.

course.

The kind of merges you're going to do are of the “one-to-many” sort: you're going to join each reacord in one DataFrame with multiple records (rows) in another DataFrame.

In addition to your custZData data, you're going to need to have the variable codes and labels in a format you can merge with what's in custZData. Let's make a couple of small DataFrames for this by putting the codes and labels into dicts, and then converting the dicts into DataFrames. Here's how to do it for the customer channel of acquisition codes:

In [1]:
```
import os

import cPickle as pickle

import pandas as pd  # panda's nickname is pd

import numpy as np  # numpy as np

import sqlalchemy

from pandas import DataFrame, Series     # for convenience
from sqlalchemy import create_engine
from sqlalchemy import schema
```

In [2]:
```
chanCode={'CHANNEL_ACQUISITION': Series(['CB','IB','RT'], index=[range(3)
          'CHAN_LAB' : Series(['Catalog','Internet','Retail'], index =[ra
```

then

In [3]:
```
chanFrame=DataFrame(chanCode)
```

should give you something that looks like:

In [4]:
```
chanFrame
```

Out[4]:

	CHANNEL_ACQUISITION	CHAN_LAB
0	CB	Catalog
1	IB	Internet
2	RT	Retail

Note that the leftmost column is the index of this DataFrame. We assigned its values using the range function, range(3), above.

Now do the same thing for the variables ZDOGS and ZCATS so that you end up with a dogFrame and a catFrame. Call the label variables in these data frames DOGHOUSE and CATHOUSE, respectively. You should end up with DataFrames that look like the following:

In [5]:
```
dogCode={'DOGHOUSE': Series(['Dog enthusiast residence','Unknown'], index
          'ZDOGS' : Series(['Y','U'], index =[range(2)])}

dogFrame=DataFrame(dogCode)

dogFrame
```

Out[5]:

	DOGHOUSE	ZDOGS
0	Dog enthusiast residence	Y
1	Unknown	U

In [6]:
```
catCode={'CATHOUSE': Series(['Cat enthusiast residence','Unknown'], index
          'ZCATS' : Series(['Y','U'], index =[range(2)])}

catFrame=DataFrame(catCode)

catFrame
```

Out[6]:

	CATHOUSE	ZCATS
0	Cat enthusiast residence	Y
1	Unknown	U

Next, let's get the variables out of custZData that we're going to include in your new json data file:

In [7]:
```
engine=create_engine('sqlite:///xyz.db')

xyzcust=pd.read_sql_table('xyzcust',engine)

zdata=pd.read_json('zdata.json',orient='index')

custZData=pd.merge(xyzcust,zdata,on='ACCTNO')

# Get the PETS now ...
custPets=custZData[['ACCTNO','CHANNEL_ACQUISITION','ZDOGS','ZCATS']]
```

custPets should have len(custZData) rows, and four columns. Don't forget to use [[and]] in the above.

How many dog and cat enthusiasts do you think there are amongst XYZ's customers? Apparently there are dog enthusiasts:

In [8]:
```
custPets.ZDOGS.value_counts()
```

Out[8]:
```
U    24730
Y     2878
Name: ZDOGS, dtype: int64
```

Are there any missing values in ZDOGS? There are:

In [9]:
```
custPets.ZDOGS.isnull().sum()
```

Out[9]:
```
2571
```

Yes, indeed, there are.

Do the above checks for ZCATS.

In [10]:
```
custPets.ZCATS.isnull().sum()
```

Out[10]:
```
2571
```

OK, so now we're ready to do some merging. We have three DataFrames, custPets, dogFrame, and catFrame. We can do these two at a time, like:

In [11]:
```
dMarketingFrame=pd.merge(custPets,dogFrame,on='ZDOGS')
```

This puts the dog information in, and:

In [12]:
```
cdMarketingFrame=pd.merge(dMarketingFrame,catFrame,on='ZCATS')
```

puts the cat information in. cdMarketingFrame is the data set we want to write out as a json file.

How many XYZ customers are both cat and dog enthusiasts?

In [13]:
```
((cdMarketingFrame.ZCATS=='Y') & (cdMarketingFrame.ZDOGS=='Y')).sum()
```

Out[13]: 786

We're going to write our custPets DataFrame out to a json file, but before we do, here's something to take a look at. Did you check to see how many rows (customers, that is), there are in cdMarketingFrame? The same number as there are in custZData, or fewer?

In [14]:
```
cdMarketingFrame.shape
```

Out[14]: (27608, 6)

In [15]:
```
cdMarketingFrame.head()
```

Out[15]:

	ACCTNO	CHANNEL_ACQUISITION	ZDOGS	ZCATS	DOGHOUSE	CATHOUSE
0	WDQQLLDQL	IB	Y	U	Dog enthusiast residence	Unknown
1	AGDDPGGQP	RT	Y	U	Dog enthusiast residence	Unknown
2	WLDAYHQLW	RT	Y	U	Dog enthusiast residence	Unknown
3	HWPPYQWS	RT	Y	U	Dog enthusiast residence	Unknown
4	APAHLSLPD	RT	Y	U	Dog enthusiast residence	Unknown

In [16]:
```
len(cdMarketingFrame)
```

Out[16]: 27608

In [17]:
```
custZData.shape
```

Out[17]: (30179, 142)

Oops. You lost some customers! Marketing won't be pleased. How did that happen? Well, there were missing data in the ZDOGS and ZCATS variables. Try:

In [18]:
```
custPets.ZCATS.value_counts(dropna=False)
```

Out[18]:
```
U      25918
NaN     2571
Y       1690
Name: ZCATS, dtype: int64
```

In [19]:
```
custPets.ZCATS.value_counts()
```

Out[19]:
```
U    25918
Y     1690
Name: ZCATS, dtype: int64
```

In [20]:
```
len(custPets)
```

Out[20]: 30179

So when you joined the DataFrames, these observations were lost. The reason is that the merge method you used was an inner join. To include all the customers in custZData in the file you send to XYZ marketing, do a left outer join instead, like:

In [21]:
```
dMarketingFrame2=pd.merge(custPets,dogFrame,on='ZDOGS', how='left')
```

In [22]:
```
cdMarketingFrame2=pd.merge(dMarketingFrame2,catFrame,on='ZCATS', how='lef
```

By the way, if your computer is beginning to respond a little more slowly to your commands, it may be because you have a lot of Python objects in RAM. You can delete those you don't need with the del command. You can delete multiple objects with a single command. Just separate their names with commas. Be careful not to delete something you need.

Now someone in marketing might reasonably ask you “Missing versus 'Unknown?'” What's the difference, really?” So if that marketing person (or perhaps your boss) insisted, you could convert those missing values to be 'U's' and join your data sets all over again. For example,

In [23]:
```
custPets.ZDOGS[custPets.ZDOGS.isnull()]='U'
```

```
/Users/Zeeshan/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages/pandas/core/generic.py:4702: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
(http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  self._update_inplace(new_data)
/Users/Zeeshan/Library/Enthought/Canopy_64bit/User/lib/python2.7/site-packages/IPython/core/interactiveshell.py:2885: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
(http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  exec(code_obj, self.user_global_ns, self.user_ns)
```

In [24]:
```
custPets[:10]
```

Out[24]:

	ACCTNO	CHANNEL_ACQUISITION	ZDOGS	ZCATS
0	WDQQLLDQL	IB	Y	U
1	WQWAYHYLA	RT	U	U
2	GSHAPLHAW	RT	U	U
3	PGGYDYWAD	RT	U	U
4	LWPSGPLLS	RT	U	U
5	LQGYDGSYQ	RT	U	None
6	WGQWQDPDA	RT	U	U
7	LPASPGYLS	RT	U	U
8	GPGDSHGL	RT	U	U
9	PQHSWQSDQ	RT	U	U

Ok, we fixed ZDOGS , we need to fix ZCATS the same way

In [25]:
```
custPets.ZCATS[custPets.ZCATS.isnull()]='U'
```

Let's now write that json file for marketing. We'll call it mktgDogsCats.json, and we'll write it to the current default directory:

In [26]:
```
cdMarketingFrame2.to_json('mktgDogsCats.json',orient='index')
```

That was easy, right? This file will be an ASCII by default. It's really just one long character string. Note that pandas can create json files with different organizations. For example, if you set orient='columns', the variable names of your DataFrame will be values on the primary index, with Series inside each index value being a column of data for that variable. If we wrote our cdMarketingFrame2 DataFrame out as json using orient='columns,' the json version would start with ACCTNO as the first value on the primary key:

'{"ACCTNO":
{"0":"WDQQLLDQL","1":"WQWAYHYLA","2":"GSHAPLHAW","3":"PGGYDYWAD","4":"LW
PSGPLLS","5":"LQGYDGSYQ","6":"WGQWQDPDA","7":"LPASPGYLS","8":"GPGDSHGL",
"9":"PQHSWQSDQ","10":"AGDDPGGQP","11":"WDSYWHW' ...

where the Series associated with ACCTNO starts with '{“0”:', the index value of the first row in cdMarketingFrame2 . So this format is essentially a long character string written by going down each column in the DataFrame, nesting Series as dicts under the variable names as values on the primary key.

Now let's do a little data type conversion just for fun. You may remember that the ZIP variable in custZData is stored as an integer:

In [27]:
```
custZData.ZIP.dtype
```

Out[27]:
```
dtype('int64')
```

Zip codes are sometimes easier to work with as character variables. And, for one thing, they are all supposed to be five digits in length. Let's get ZIP out of custZData and examine it.

In [28]:
```
ZIP=custZData.ZIP.astype('string')  #convert ZIP to character string type

type(ZIP)
```

Out[28]: pandas.core.series.Series

ZIP is now a character variable, a “string” variable.

In [29]:
```
ZIPlen=ZIP.str.len()  # ZIPlen is a Series containing zip code string len
```

In [30]:
```
ZIPlen.value_counts()  # what are the various lengths of ZIP?
```

Out[30]:
```
5    30178
1        1
Name: ZIP, dtype: int64
```

It appears that there is one ZIP code with a single digit. What is that digit?

In [31]:
```
ZIP[(ZIPlen==1)]
```

Out[31]:
```
19337    0
Name: ZIP, dtype: object
```

The single digit is zero, and its in the row with an index value of 19337. (You may recall that a few Sessions ago it was mentioned that missing values in ZIP, ZIP4, and ZIP9_SUPERCODE were coded as as zero.

We can change that zero to a missing value in custZData:

In [32]:
```
custZData.loc[ZIPlen==1,'ZIP']=None
```

The zero has now been replaced by NaN

In [33]:
```
custZData.loc[ZIPlen==1,'ZIP']
```

Out[33]:
```
19337   NaN
Name: ZIP, dtype: float64
```

Note that the data type is still 64 bit float. We created ZIP as a Series of character strings outside of the DataFrame custZData. So ZIP in custZData remains unchanged. We could add our Series ZIP to cust if we wanted to, after setting that zero to None in it:

In [34]:
```
ZIP[ZIPlen==1]=None
```

In [35]:
```
custZData['sZIP']=ZIP
```

We can do the same thing to ZIP4 and ZIP9_SUPERCODE, of course. Note that ZIP4 should be 4 digits, but if it's stored as an integer, leading zeros may have disappeared from it. Let's take a look:

In [36]:
```
ZIP4=custZData.ZIP4  # make a ZIP4 series
ZIP4=ZIP4.astype('string')  # make the series a series of character strin
ZIP4len=ZIP4.str.len()  #  ZIP4len is a Series of string lengths
ZIP4len.value_counts(dropna=False) # What are the different string length
```

Out[36]:
```
4    26332
1     3616
3      218
2       13
Name: ZIP4, dtype: int64
```

Egad! There's a bunch that have fewer than 4 characters! (We hope that what they all have are digits, right?) Let's take a closer look at those with less than four, starting with those with only 2:

In [37]:
```
ZIP4.loc[ZIP4len==2]
```

Out[37]:
```
51       24
1633     20
2567     45
2803     68
4219     66
6012     97
15290    33
16916    54
21762    70
21855    41
26952    10
27157    68
28008    84
Name: ZIP4, dtype: object
```

Note that the left column above has the index values for the values returned from ZIP4. the righthand column are the ZIP4 digits. If left-padded with zeros, these would probably be valid ZIP4 values.

There are a lot of ZIP4's with a length of 1. You'll recall that missing values were coded as zero. So we might want to set those that are zero equal to None. To fix these ZIP4's up to have four digits, we might proceed as follows:

might proceed as follows:

(1) Pad all ZIP4's with fewer than 4 characters with zero characters on the left so that they have four characters.
(2) Set ZIP4's that are four zeros, "0000," to None.

Does this plan make sense? Any weaknesses? We'll work on it in the next Session's Practice.

Be sure to save everything that's important. You may as well save ZIP, ZIPlen, ZIP4, and ZIP4len, as you may find them handy in the next Session Practice.

If you want to pickle them, you can put them into a DataFrame and then pickle the whole thing, like this:

In [38]:
```
zipVars=[ZIP,ZIPlen,ZIP4,ZIP4len]   # a list of Series
```

In [39]:
```
zipFrame=pd.concat(zipVars,axis=1)  # put Series side by side in zipFrame
```

In [40]:
```
zipFrame.shape          #dimensions, please?
```

Out[40]: (30179, 4)

In [41]:
```
zipFrame.head()
```

Out[41]:

	ZIP	ZIP	ZIP4	ZIP4
0	60084	5	5016	4
1	60091	5	1750	4
2	60067	5	900	3
3	60068	5	3838	4
4	60090	5	3932	4

That looks ok, doesn't it? And the columns are:

In [42]:
```
zipFrame.columns
```

Out[42]: Index([u'ZIP', u'ZIP', u'ZIP4', u'ZIP4'], dtype='object')

So now let's pickle this Frame to your working directory:

In [43]:
```
zipFrame.to_pickle('pickledZIPs.p')
```

```
pandas DataFrames and Series have a pickle method that uses cPickle.
```

Deliverable #1: Print a list of ZIP codes and their frequences in ZIP

In [44]:
```
ZIP.value_counts()
```

Out[44]:
```
60091    3422
60093    3152
60062    3059
60067    3022
60068    2754
60089    1990
60056    1515
60074    1297
60060    1283
60061    1197
60076    1081
60069     776
60077     735
60084     717
60073     678
60090     644
60098     562
60070     458
60085     377
60083     342
60081     318
60087     264
60097     150
60096     122
60071      97
60064      41
60072      34
60088      28
60078      25
60065      21
60075       5
60094       4
60082       3
60192       2
60079       2
60095       1
Name: ZIP, dtype: int64
```

Deliverable #2: how many cat enthusiasts?

In [45]:
```
CatEnthusiasts = custPets.ZCATS[custPets.ZCATS=="Y"]
CatEnthusiasts.value_counts()
```

Out[45]:
```
Y    1690
Name: ZCATS, dtype: int64
```

Deliverable #3: Are there any missing values in ZDOGS?

In [46]:
```
custPets.ZDOGS.isnull().sum()
```

Out[46]: 0