pandas.DataFrame to ArcGIS Table

Pandas is an incredibly convenient Python module for working with tabular data when ArcGIS table tools and workflows are missing functionality or are simply too slow. Panda's main data structure, the DataFrame, cannot be directly ingested back into a GDB table. Esri's tool to do this, NumPyArrayToTable(), only reads numpy arrays. More specifically, the tool requires a *structured* numpy array, which means that each column needs to have a "dtype" definition that specifies the name and data type (like "int16").

There are several problems, the first of which is that the conversion from the pandas.DataFrame to a numpy.array tends to strip off the specification of the data type in each column, or at least it does *some* of the things you want, but not all. When making a pandas-->numpy conversion, each column is cast from a specific pandas data type to a corresponding numpy data type. The crux of the problem is that there are not equivalent numpy types for all pandas data types (most, but definitely not all). In these cases, the conversion "upcasts" to a more generic data type. This frequently results in an output data type of "object", which is so generic that the arcpy.da.NumPyArrayToTable() function barfs. This conversion to "object" happens for string fields, for example. If the pandas.DataFrame columns have different data types, then the conversion usually just treats all columns as "object".

Here is an example of how to deal with this. First, the sample pandas.DataFrame:

source data

```
[Dbg]>>> type(chorizonVar)
<class 'pandas.core.frame.DataFrame'>

[Dbg]>>> chorizonVar.dtypes
cokey              object
chkey              object
hzdept_r            int64
hzdepb_r            int64
claytotal_r       float64
hzdepb_r20          int64
h_thk               int64
co_thk            float64
co_wt             float64
claytotal_r_wt    float64
```

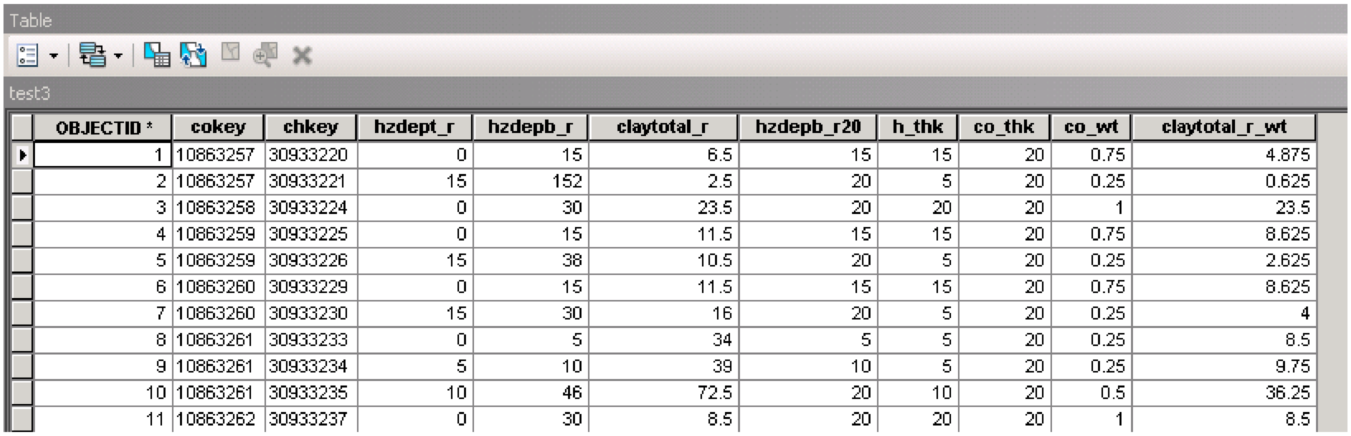
There are a variety functions within both pandas and numpy that can convert a pandas.DataFrame to a numpy.array, so there might be additional methods for doing this. Here is one example.

source data

```
x = np.array(np.rec.fromrecords(chorizonVar.values))
names = chorizonVar.dtypes.index.tolist()
x.dtype.names = tuple(names)
arcpy.da.NumPyArrayToTable(x, r'E:\Workspace\testData.gdb\testTable')
```

Most of the action/smarts is/are happening on Line 1. The first thing is the conversion of the pandas.DataFrame to an **unstructured** numpy.array with *chorizonVar.values*. This np.array characterizes all columns as "object". The real magic is turning this into a numpy.recarray (aka "record array") with *np.rec.fromrecords()*. This function looks at the content of each column and makes its best guess, which worked ideally in my experiment that contained strings, integers, and floats. The final thing that happens on this line is converting from a np.recarray in to a regular np.array because that is what's required for use with *arcpy.da.NumPyArrayToTable().* Also worth noting that while the np.recarray data structure is generally a "richer" data structure than an np.array, it is also substantially more expensive in terms of memory and the speed of operations applied to it.

So, now we have a numpy.array, *x*, with good data type specifications. The only problem is that the names of the columns have been dumbed down to "f0", "f1" and so on (for "field 0", etc). Since I want to keep the names that were in the pandas.DataFrame, I pull those into a list on Line 2 and then reset the names in the numpy.array with Line 3. The numpy.array is now just what I want to go into the ArcGIS gdb table (Line 4). The first few rows of the result is shown below.



OBJECTID *	cokey	chkey	hzdept_r	hzdepb_r	claytotal_r	hzdepb_r20	h_thk	co_thk	co_wt	claytotal_r_wt
1	10863257	30933220	0	15	6.5	15	15	20	0.75	4.875
2	10863257	30933221	15	152	2.5	20	5	20	0.25	0.625
3	10863258	30933224	0	30	23.5	20	20	20	1	23.5
4	10863259	30933225	0	15	11.5	15	15	20	0.75	8.625
5	10863259	30933226	15	38	10.5	20	5	20	0.25	2.625
6	10863260	30933229	0	15	11.5	15	15	20	0.75	8.625
7	10863260	30933230	15	30	16	20	5	20	0.25	4
8	10863261	30933233	0	5	34	5	5	20	0.25	8.5
9	10863261	30933234	5	10	39	10	5	20	0.25	9.75
10	10863261	30933235	10	46	72.5	20	10	20	0.5	36.25
11	10863262	30933237	0	30	8.5	20	20	20	1	8.5