Creating a Pandas DataFrame with a numpy array containing multiple types

I want to create a pandas dataframe with default values of zero, but one column of integers and the other of floats. I am able to create a numpy array with the correct types, see the `values` variable below. However, when I pass that into the dataframe constructor, it only returns NaN values (see `df` below). I have include the untyped code that returns an array of floats(see `df2`)

```
import pandas as pd
import numpy as np

values = np.zeros((2,3), dtype='int32,float32')
index = ['x', 'y']
columns = ['a','b','c']

df = pd.DataFrame(data=values, index=index, columns=columns)
df.values.dtype

values2 = np.zeros((2,3))
df2 = pd.DataFrame(data=values2, index=index, columns=columns)
df2.values.dtype
```

Any suggestions on how to construct the dataframe?

python numpy pandas

asked Feb 8 '14 at 14:12

bfcondon
49 1 7

1 Answer

Here are a few options you could choose from:

```
import numpy as np
import pandas as pd

index = ['x', 'y']
columns = ['a','b','c']

# Option 1: Set the column names in the structured array's dtype
dtype = [('a','int32'), ('b','float32'), ('c','float32')]
values = np.zeros(2, dtype=dtype)
df = pd.DataFrame(values, index=index)

# Option 2: Alter the structured array's column names after it has been created
values = np.zeros(2, dtype='int32, float32, float32')
values.dtype.names = columns
df2 = pd.DataFrame(values, index=index, columns=columns)

# Option 3: Alter the DataFrame's column names after it has been created
values = np.zeros(2, dtype='int32, float32, float32')
df3 = pd.DataFrame(values, index=index)
df3.columns = columns

# Option 4: Use a dict of arrays, each of the right dtype:
df4 = pd.DataFrame(
    {'a': np.zeros(2, dtype='int32'),
     'b': np.zeros(2, dtype='float32'),
     'c': np.zeros(2, dtype='float32')}, index=index, columns=columns)

# Option 5: Concatenate DataFrames of the simple dtypes:
df5 = pd.concat([
    pd.DataFrame(np.zeros((2,), dtype='int32'), columns=['a']),
    pd.DataFrame(np.zeros((2,2), dtype='float32'), columns=['b','c'])], axis=1)

# Option 6: Alter the dtypes after the DataFrame has been formed. (This is not very
efficient)
values2 = np.zeros((2, 3))
df6 = pd.DataFrame(values2, index=index, columns=columns)
```

```
for col, dtype in zip(df6.columns, 'int32 float32 float32'.split()):
    df6[col] = df6[col].astype(dtype)
```

Each of the options above produce the same result

```
   a  b  c
x  0  0  0
y  0  0  0
```

with dtypes:

```
a      int32
b    float32
c    float32
dtype: object
```

Why `pd.DataFrame(values, index=index, columns=columns)` produces a DataFrame with NaNs:

`values` is a structured array with column names `f0` , `f1` , `f2` :

```
In [171]:  values
Out[172]:
array([(0, 0.0, 0.0), (0, 0.0, 0.0)],
      dtype=[('f0', '<i4'), ('f1', '<f4'), ('f2', '<f4')])
```

If you pass the argument `columns=['a', 'b', 'c']` to `pd.DataFrame` , then Pandas will look for columns with those names in the structured array `values` . When those columns are not found, Pandas places `NaN` s in the DataFrame to represent missing values.

edited Sep 15 '15 at 17:56

answered Feb 8 '14 at 14:25

unutbu
334k 38 583 722

It would be nice to know why this works, so we don't just copy and paste the solution. Thanks! – rocarvaj Sep 15 '15 at 13:43

@rocarvaj: What is it that you feel needs expalnation? – unutbu Sep 15 '15 at 14:57

When to use the standard DataFrame constructor and when to use from_records. – rocarvaj Sep 15 '15 at 15:02

1 @rocarvaj: I don't think my original solution, using `pd.DataFrame.from_records` , is a good option because it does not produce a DataFrame with the desired column names. So I've rewritten my answer to show other alternatives. – unutbu Sep 15 '15 at 18:01

@rocarvaj: I don't know of a situation where using `pd.DataFrame.from_records` is more convenient than using `pd.DataFrame` itself. – unutbu Sep 15 '15 at 18:08