

Bridging the pandas — scikit-learn dtype divide

Tom Augspurger

The Problem

Two Data Models

	Broadcast- ing	Vectorization	ufuncs	typed arrays	ND-arrays	Labels	Heterogeneous	Extension dtypes
scikit- learn / NumPy	✓	✓	✓	✓	✓			
Pandas	✓	✓	✓	✓		✓	✓	✓

Two Data Models

Claim: “Real world” data are

- Labeled
- Heterogenous
- Messy

The Data

```
In [1]: import numpy as np
...: import pandas as pd
...: import seaborn as sns
...:
...: tips = pd.read_csv('tips.csv')
...: tips.head()
...:
```

Out[1]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

The Data

```
In [2]: tips.info()
```

```
Out[2]:
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 244 entries, 0 to 243
```

```
Data columns (total 7 columns):
```

```
total_bill    244 non-null float64
```

```
tip           244 non-null float64
```

```
sex           244 non-null object
```

```
smoker        244 non-null object
```

```
day           244 non-null object
```

```
time          244 non-null object
```

```
size          244 non-null int64
```

```
dtypes: float64(2), int64(1), object(4)
```

```
memory usage: 13.4+ KB
```

```
In [3]: X = tips.drop("tip", axis=1)
```

```
...: y = tips["tip"]
```

The Stats

The Statistics: linear model

```
>>> model = LinearRegression()
```

```
>>> model.fit(X, y)
```


The Statistics: linear model

```
>>> model = LinearRegression()
```

```
>>> model.fit(X, y)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
File "sklearn/linear_model/base.py", line 427, in fit
```

```
    y_numeric=True, multi_output=True)
```

```
File "sklearn/utils/validation.py", line 510, in check_X_y
```

```
    ensure_min_features, warn_on_dtype, estimator)
```

```
File "sklearn/utils/validation.py", line 393, in check_array
```

```
    array = array.astype(np.float64)
```

```
ValueError: could not convert string to float: 'Dinner'
```

The Statistics: linear model

```
>>> model = LinearRegression()
```

```
>>> model.fit(X, y)
```

```
Traceback (most recent call last):
```

```
File "<stdin>", line 1, in <module>
```

```
File "sklearn/linear_model/base.py", line 427, in fit
```

```
    y_numeric=True, multi_output=True)
```

```
File "sklearn/utils/validation.py", line 510, in check_X_y
```

```
    ensure_min_features, warn_on_dtype, estimator)
```

```
File "sklearn/utils/validation.py", line 393, in check_array
```

```
    array = array.astype(np.float64)
```

```
ValueError: could not convert string to float: 'Dinner'
```

$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T \mathbf{X}\right)^{-1} \mathbf{X}^T \mathbf{y}$$

Aside: R vs. Python

```
> lm(tip ~ ., tips)
```

Call:

```
lm(formula = tip ~ ., data = tips)
```

Coefficients:

(Intercept)	total_bill	sexMale	smokerYes	daySat	daySun
0.80382	0.09449	-0.03244	-0.08641	-0.12146	-0.02548
dayThur	timeLunch	size			
-0.16226	0.06813	0.17599			

transformations

transformations: `pd.factorize`

Raw	Factorized
	0
	0
	1
	2
	2
	3
	1
	3
	0
	2
	0
	3

transformations: `pd.factorize`

```
In [4]: codes, labels = pd.factorize(tips['day'])
```

```
....: codes
```

```
....:
```

```
Out[4]:
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,  
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,  
       ...,  
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2])
```

Raw



Factorized

0
0
1
2
2
3
1
3
0
2
0
3

transformations: `pd.factorize`

```
In [5]: X_factorized = X.copy()
...: columns = ['sex', 'smoker', 'day', 'time']
...: X_factorized[columns] = X[columns].apply(lambda x: pd.factorize(x)[0])
...:
...: lm = LinearRegression()
...: lm.fit(X_factorized, y)
...: lm.coef_
```

Out[5]:

```
array([ 0.09407595, -0.02921608, -0.08104051, -0.00783382,  0.00572054,
        0.17936677])
```

transformations: `pd.factorize`

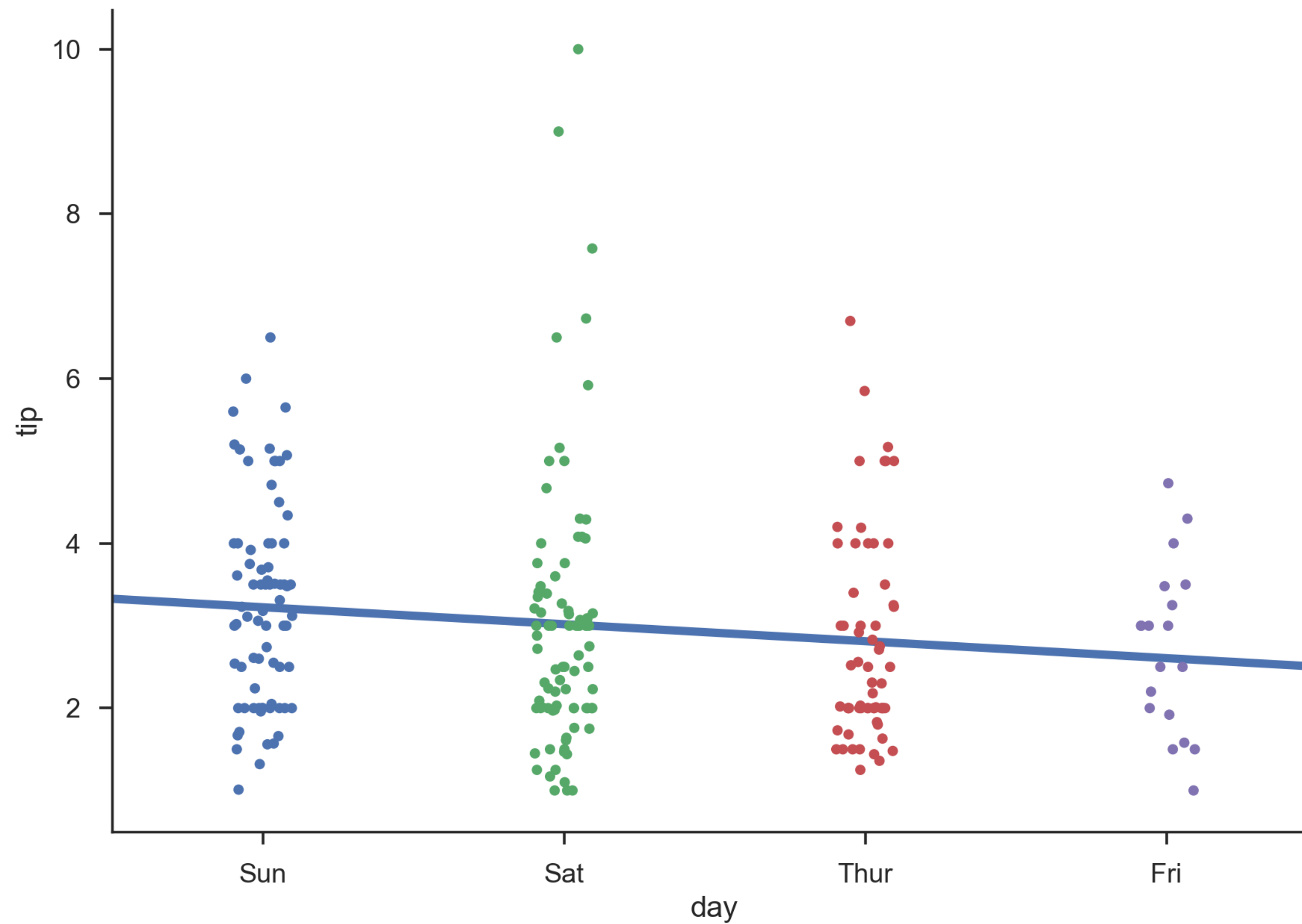
That “worked”, but

- Ordering becomes important
- All categories are equally spaced
- All categories have equal effects^{*}

$$\frac{\Delta \text{tip}}{\Delta(\text{Sun.} \rightarrow \text{Sat.})} = \frac{\Delta \text{tip}}{\Delta(\text{Thur.} \rightarrow \text{Fri.})}$$

* for our linear model

transformations: `pd.factorize`



transformations: Dummy Encoding

Raw	D1	D2	D3	D4
	1	0	0	0
	1	0	0	0
	0	1	0	0
	0	0	1	0
	0	0	1	0
	0	0	0	1
	0	1	0	0
	0	0	0	1
	1	0	0	0
	0	0	1	0
	1	0	0	0
	0	0	0	1

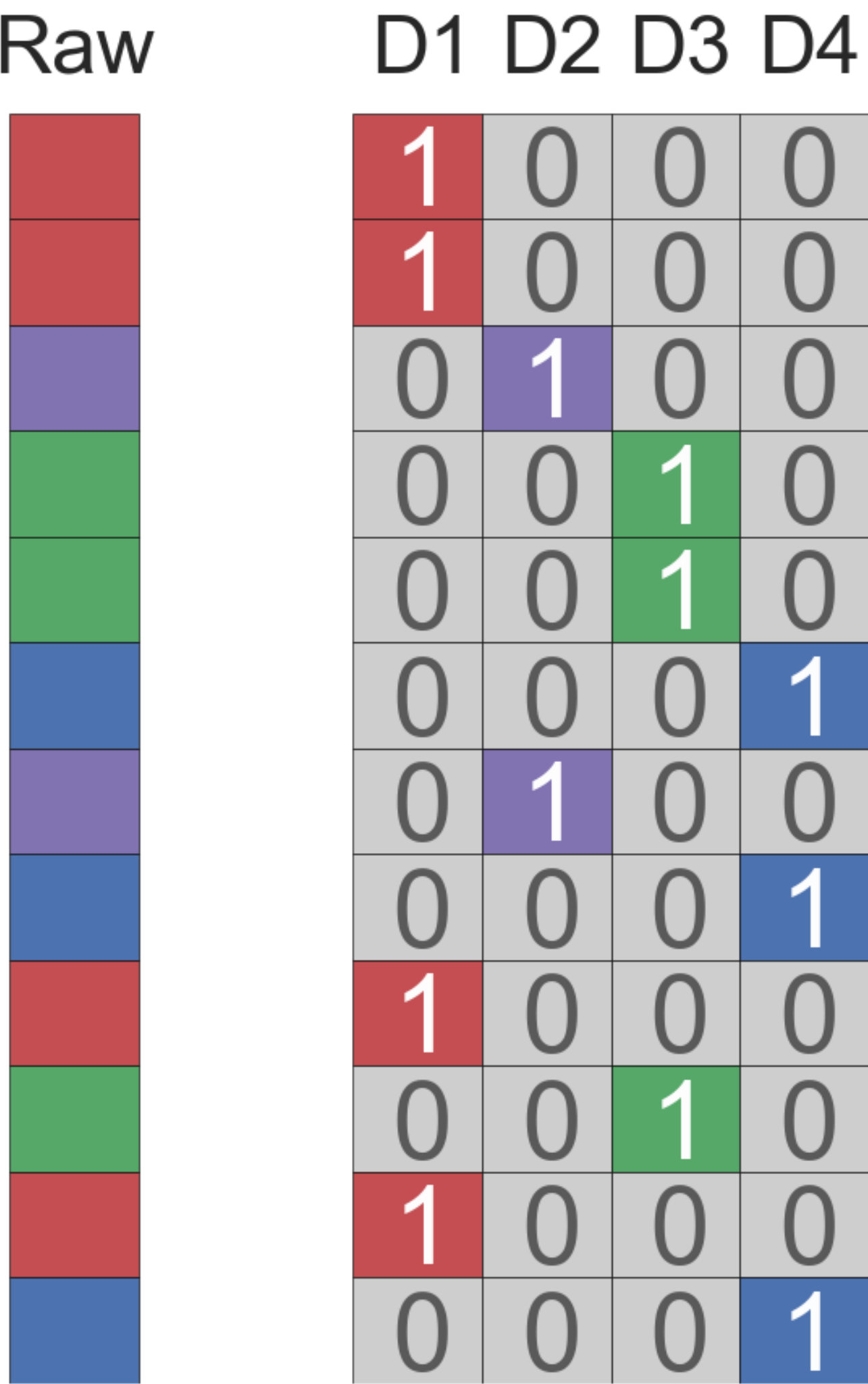
transformations: Dummy Encoding

```
In [6]: pd.get_dummies(df.day)
```

Out[6]:

	Fri	Sat	Sun	Thur
0	0.0	0.0	1.0	0.0
1	0.0	0.0	1.0	0.0
2	0.0	0.0	1.0	0.0
3	0.0	0.0	1.0	0.0
4	0.0	0.0	1.0	0.0
..
239	0.0	1.0	0.0	0.0
240	0.0	1.0	0.0	0.0
241	0.0	1.0	0.0	0.0
242	0.0	1.0	0.0	0.0
243	0.0	0.0	0.0	1.0

[244 rows x 4 columns]



transformations: pd.get_dummies

```
In [7]: X_dummies = pd.get_dummies(X)
...: X_dummies.head()
```

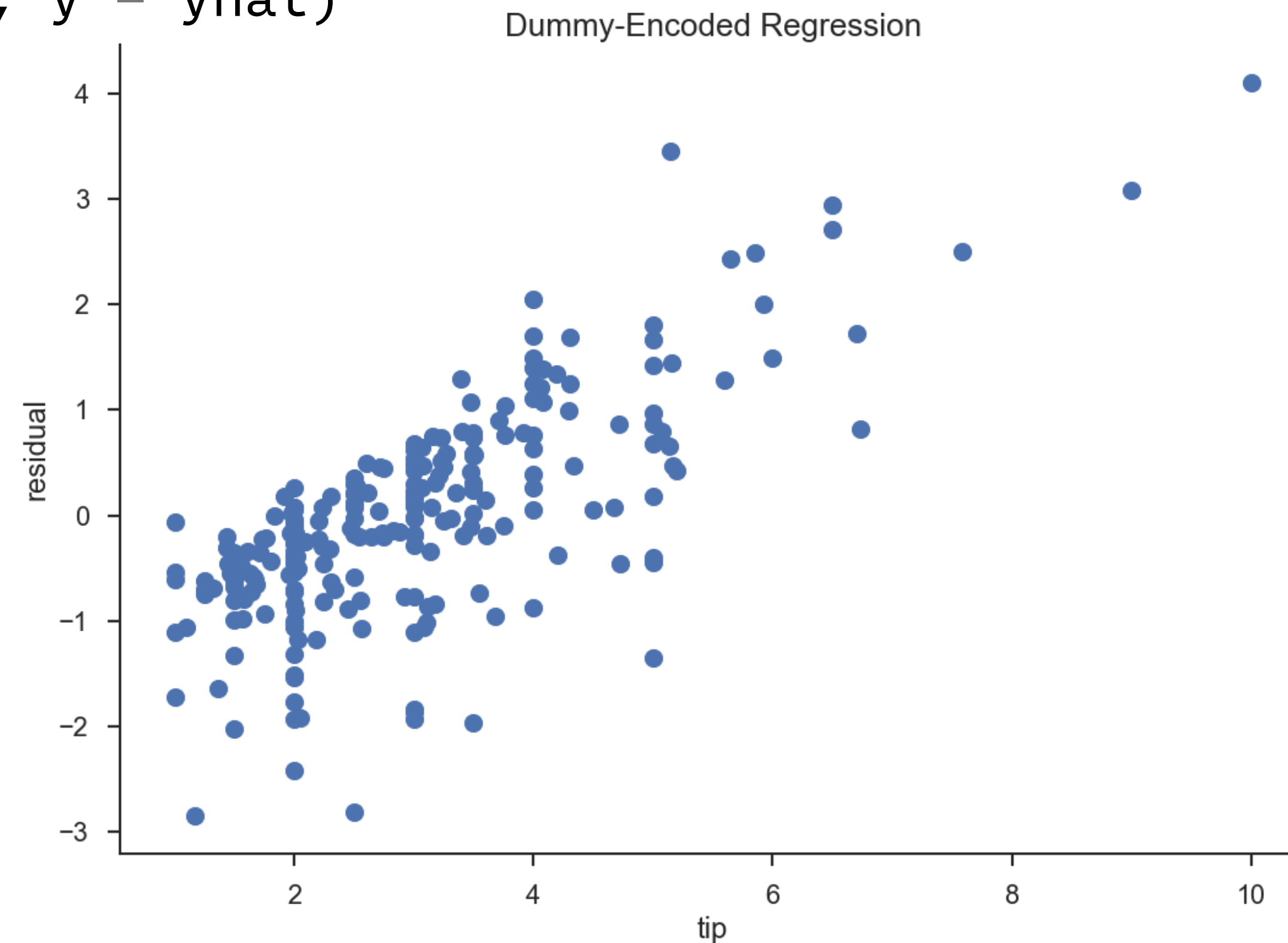
```
Out[7]:
```

	total_bill	size	sex_Female	sex_Male	smoker_No	smoker_Yes	day_Fri	\
0	16.99	2	1.0	0.0	1.0	0.0	0.0	
1	10.34	3	0.0	1.0	1.0	0.0	0.0	
2	21.01	3	0.0	1.0	1.0	0.0	0.0	
3	23.68	2	0.0	1.0	1.0	0.0	0.0	
4	24.59	4	1.0	0.0	1.0	0.0	0.0	

	day_Sat	day_Sun	day_Thur	time_Dinner	time_Lunch
0	0.0	1.0	0.0	1.0	0.0
1	0.0	1.0	0.0	1.0	0.0
2	0.0	1.0	0.0	1.0	0.0
3	0.0	1.0	0.0	1.0	0.0
4	0.0	1.0	0.0	1.0	0.0

transformations: `pd.get_dummies`

```
In [8]: from sklearn.linear_model import LinearRegression
...: lm = LinearRegression().fit(X_dummies, y)
...:
...: yhat = lm.predict(X_dummies)
...: plt.scatter(y, y - yhat)
```



DummyEncoder

DummyEncoder

The last solution is brittle

1. We can't easily go from dummies back to categoricals
2. Doesn't integrate with scikit-learn `Pipeline` objects.
3. If working with a larger dataset and `partial_fit`, codes could be missing from subsets of the data.
4. Memory inefficient if there are many records relative to distinct categories

DummyEncoder

Step 1: Categorize

```
In [9]: columns = ['sex', 'smoker', 'day', 'time']
...: tips[columns] = tips[columns].apply(lambda x: x.astype('category'))
...: tips.info()
```

```
In [9]: tips.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 7 columns):
total_bill    244 non-null float64
tip           244 non-null float64
sex           244 non-null category
smoker        244 non-null category
day           244 non-null category
time          244 non-null category
size          244 non-null int64
dtypes: category(4), float64(2), int64(1)
memory usage: 6.8 KB
```

Stores values information in the type

- Categories
- Order

DummyEncoder

Step 2: Transformer

DummyEncoder

```
import numpy as np
import pandas as pd
from sklearn.pipeline import TransformerMixin

class DummyEncoder(TransformerMixin):
    ...
```

DummyEncoder

```
class DummyEncoder(TransformerMixin):

    def fit(self, X, y=None):
        self.index_ = X.index
        self.columns_ = X.columns
        self.cat_columns_ = X.select_dtypes(include=['category']).columns
        self.non_cat_columns_ = X.columns.drop(self.cat_columns_)

        self.cat_map_ = {col: X[col].cat for col in self.cat_columns_}

        left = len(self.non_cat_columns_)
        self.cat_blocks_ = {}
        for col in self.cat_columns_:
            right = left + len(X[col].cat.categories)
            self.cat_blocks_[col], left = slice(left, right), right
        return self
```

DummyEncoder

```
class DummyEncoder(TransformerMixin):  
    ...  
  
    def transform(self, X, y=None):  
        return np.asarray(pd.get_dummies(X))
```

DummyEncoder

```
class DummyEncoder(TransformerMixin):  
    ...  
  
    def inverse_transform(self, X):  
        non_cat = pd.DataFrame(X[:, :len(self.non_cat_columns_)],  
                                columns=self.non_cat_columns_)  
  
        cats = []  
        for col, cat in self.cat_map_.items():  
            slice_ = self.cat_blocks_[col]  
            codes = X[:, slice_].argmax(1)  
            series = pd.Series(pd.Categorical.from_codes(  
                codes, cat.categories, ordered=cat.ordered  
            ), name=col)  
            cats.append(series)  
        df = pd.concat([non_cat] + cats, axis=1)[self.columns_]   
        return df
```

DummyEncoder

```
In [14]: X = tips.drop('tip', axis=1)
...: y = tips['tip']
...:
...: pipe = make_pipeline(
...:     DummyEncoder(),
...:     LinearRegression()
...: )
...: pipe.fit(X, y)
```

```
Out[14]: Pipeline(
  steps=[('dummyencoder',
    <DummyEncoder object at 0x10992af60>),
    ('linearregression',
    LinearRegression(copy_X=True, fit_intercept=True,
      n_jobs=1, normalize=False))])
```

Takeaway

Takeaway

- That particular transformer isn't the point
- Pandas is evolving further, more bridges needed
- Be comfortable writing the shims (or write a library to do it?)

Takeaway

<https://mail.python.org/mailman/listinfo/pandas-dev>

<https://github.com/pydata/pandas>

