Using the `apply()` method in pandas

Sometimes, creating a calculated column in pandas is as simple as this:

```
df['difference'] = df['first_column'] - df['second_column']
```

or this:

```
df['date_fixed'] = pd.to_datetime(df['date'])
```

Other times, though, your needs are more complex -- you need to take each row of data in your data frame and do *several things* to it. That's where `apply()` comes in.

Given a function, `apply()` will, uh, *apply* that function to every row in the data frame. A common scenario for doing so would be to create a new column.

An example might make this idea a little more clear. Let's load up a CSV of Texas death row media witnesses.

In [16]:
```
import pandas as pd
```

In [17]:
```
df = pd.read_csv('../data/tx-death-row-media-list.csv', parse_dates=['execution_date'])
```

Now, let's say, we want to create a new column with the *month* of the execution. Given what we know about date objects, this should be simple, right?

So this might be my first guess:

In [18]:
```
df['month'] = df['execution_date'].month
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
/var/folders/6p/3792ml551vv6d6c6yvwm5py00000gn/T/ipykernel_84144/2562246086.py in <module>
----> 1 df['month'] = df['execution_date'].month

~/ire/custom-training/jni-intermediate-2021/env/lib/python3.9/site-packages/pandas/core/ge
neric.py in __getattr__(self, name)
   5485         ):
   5486             return self[name]
-> 5487         return object.__getattribute__(self, name)
   5488
   5489     def __setattr__(self, name: str, value) -> None:

AttributeError: 'Series' object has no attribute 'month'
```

Womp womp. Looks like we need to create a *function* to do this for us. Then we can *apply* that function to each row.

👉 For a refresher on writing your own functions, check out this notebook.

In [19]:
```
def get_month(row):
    '''Given a row of data, return the month of the execution date'''
    return row['execution_date'].month
```

... and now we can apply it. We also need to specify *how* it's going to be applied. `axis=0` is the default and attempts to apply the function to each *column*. We want `axis=1` , which applies the function to each *row* of data.

In [20]:

```
df['month'] = df.apply(get_month, axis=1)
```

In [21]:

```
df.head()
```

Out[21]:

	execution_no	execution_date	journo_last	journo_rest	journo_affiliation	inmate_no	inmate_last	inmate_
0	572	2021-06-30	Graczyk	Michael	Associated Press	999567	Hummel	
1	572	2021-06-30	Brown	Joseph	Huntsville Item	999567	Hummel	
2	571	2021-05-19	No media witnesses present.	NaN	NaN	999379	Jones	Qu
3	570	2020-07-08	Graczyk	Michael	Associated Press	999137	Wardlow	
4	570	2020-07-08	Brown	Joseph	Huntsville Item	999137	Wardlow	

We could also have dropped in a *lambda expression* for the function -- in this case, it's simple enough to be readable:

In [22]:

```
df['month'] = df.apply(lambda x: x['execution_date'].month, axis=1)
```