dtype

October 7, 2018

1 Dtype

1.1 Data Type Object

Let's look into how you might generate positions from signals. To do that, we first need to know about `dtype` or data type objects in Numpy.

A data type object is a class that represents the data. It's similar to a data type, but contains more information about the data. Let's see an example of a data type object in Numpy using the array `array`.

```
In [1]: import numpy as np

        array = np.arange(10)

        print(array)
        print(type(array))
        print(array.dtype)
```

```
[0 1 2 3 4 5 6 7 8 9]
<class 'numpy.ndarray'>
int64
```

From this, we see `array` is a `numpy.ndarray` with the data `[0 1 2 3 4 5 6 7 8 9]` represented as `int64` (64-bit integer).

Let's see what happens when we divide the data by 2 to generate not integer data.

```
In [2]: float_arr = array / 2

        print(float_arr)
        print(type(float_arr))
        print(float_arr.dtype)
```

```
[ 0.   0.5  1.   1.5  2.   2.5  3.   3.5  4.   4.5]
<class 'numpy.ndarray'>
float64
```

The array returned has the values `[ 0.   0.5  1.   1.5  2.   2.5  3.   3.5  4.   4.5]`, which is what you would expect for divinding by 2. However, since this data can't be represeted by integers, the array is now represented as `float64` (64-bit float).

How would we convert this back to `int64`? We'll use the `ndarray.astype` function to cast it from it's current type to the type of `int64` (`np.int64`).

```
In [3]: int_arr = float_arr.astype(np.int64)

        print(int_arr)
        print(type(int_arr))
        print(int_arr.dtype)
```

```
[0 0 1 1 2 2 3 3 4 4]
<class 'numpy.ndarray'>
int64
```

This casts the data to `int64`, but all also changes the data. Since fractions can't be represented as integers, the decimal place is dropped.

1.2 Signals to Positions

Now that you've seen how the a data type object is used in Numpy, let's see how to use it to generate positions from signals. Let's use `prices` array to represent the prices in dollars over time for a single stock.

```
In [4]: prices = np.array([1, 3, -2, 9, 5, 7, 2])

        prices
```

```
Out[4]: array([ 1,  3, -2,  9,  5,  7,  2])
```

For the positions, let's say we want to buy one share of stock when the price is above 2 dollars and the buy 3 more shares when it's above 4 dollars. We'll first need to generate the signal for these two positions.

```
In [5]: signal_one = prices > 2
        signal_three = prices > 4

        print(signal_one)
        print(signal_three)
```

```
[False  True False  True  True  True False]
[False False False  True  True  True False]
```

This gives us the points in time for the signals above 2 dollars and above 4 dollars. To turn this into positions, we need to multiply each array by the respective amount to invest. We first need to turn each signal into an integer using the `ndarray.astype` function.

```
In [6]: signal_one = signal_one.astype(np.int)
        signal_three = signal_three.astype(np.int)

        print(signal_one)
        print(signal_three)
```

```
[0 1 0 1 1 1 0]
[0 0 0 1 1 1 0]
```

Now we multiply each array by the respective amount to invest.

```
In [7]: pos_one = 1 * signal_one
        pos_three = 3 * signal_three

        print(pos_one)
        print(pos_three)
```

```
[0 1 0 1 1 1 0]
[0 0 0 3 3 3 0]
```

If we add them together, we have the final position of the stock over time.

```
In [8]: long_pos = pos_one + pos_three

        print(long_pos)
```

```
[0 1 0 4 4 4 0]
```

1.3 Quiz

Using this information, implement `generate_positions` using Pandas's `df.astype` function to convert `prices` to final positions using the following signals: - Long 30 share of stock when the price is above 50 dollars - Short 10 shares of stock when it's below 20 dollars

```
In [9]: import project_tests
        import pandas as pd

        def generate_positions(prices):
            """
            Generate the following signals:
             - Long 30 share of stock when the price is above 50 dollars
             - Short 10 shares when it's below 20 dollars

            Parameters
            ----------
            prices : DataFrame
        Prices for each ticker and date

    Returns
    -------
    final_positions : DataFrame
        Final positions for each ticker and date
    """
     # TODO: Implement Function
    signal_long = (prices > 50).astype(np.int)
    signal_short = (prices < 20).astype(np.int)

    pos_long = 30 * signal_long
    pos_short = -10 * signal_short

    return pos_long + pos_short


project_tests.test_generate_positions(generate_positions)
```

```
Tests Passed
```

1.4 Quiz Solution

If you're having trouble, you can check out the quiz solution here.