Looping using the .iterrows() function

OPTIMIZING PYTHON CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

The poker dataset

	S1	R1	S2	R2	S3	R3	S4	R4	S5	R5
1	1	10	3	11	3	13	4	4	2	1
2	2	11	2	13	2	10	2	12	2	1
3	3	12	3	11	3	13	3	10	3	1

1. Hearts
2. Diamonds
3. Clubs
4. Spades

Generators in Python

```
def city_name_generator():
    yield('New York')
    yield('London')
    yield('Tokyo')
    yield('Sao Paolo')

city_names = city_name_generator()
```

```
next(city_names)
'New York'
next(city_names)
'London'
next(city_names)
'Tokyo'
next(city_names)
'Sao Paolo'
```

```
next(city_names)
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```

Looping using the .iterrows() function

```
gen = poker.iterrows()
first_element = next(gen)
```

```
first_element[0]
0
```

```
first_element[1]
S1      2
R1     11
S2      2
R2     13
S3      2
R3     10
S4      2
R4     12
S5      2
R5      1
Name: 1, dtype: int64
```

Using the .iterrows() function

```
start_time = time.time()
for index, values in range(poker.shape[0]):
    next
print("Time using range(): {} sec".format(time.time() - start_time))
```

```
Results using range(): 0.006870031 sec
```

```
data_generator = poker.iterrows()

start_time = time.time()
for index, values in data_generator:
    next
print("Time using .iterrows(): {} sec".format(time.time() - start_time))
```

```
Time using .iterrows(): 1.55003094673 sec
```

Let's do it!

OPTIMIZING PYTHON CODE WITH PANDAS

Looping using the .apply() function

OPTIMIZING PYTHON CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

The .apply() function

```
data_sqrt = poker.apply(lambda x: np.sqrt)
head(data_sqrt, 4)
```

	S1	R1	S2	R2	S3	R3
0	1.000000	3.162278	1.000000	3.316625	3.464102	1.000000
1	1.414214	3.316625	1.414214	3.605551	1.414214	3.162278
2	1.732051	3.464102	1.732051	3.316625	1.732051	3.605551
3	2.000000	3.162278	2.000000	3.316625	2.000000	1.000000

```
data_sqrt_2 = np.sqrt(poker)
```

The .apply() function for rows

```
apply_start_time = time.time()
poker[['R1', 'R2', 'R3', 'R4', 'R5']].apply(lambda x: sum(x), axis=1)
print("Time using .apply(): {} sec".format(time.time() - apply_start_time))
```

```
Time using .apply(): 0.636334896088 sec
```

```
start_time = time.time()
for ind, value in poker.iterrows():
    sum([value[1], value[3], value[5], value[7], value[9]])
print("Time using .iterrows(): {} sec".format(time.time() - start_time))
```

```
Time using .iterrows(): 3.15526986122 sec
```

```
Difference in speed: 395.85051529%
```

The .apply() function for columns

```
start_time = time.time()
poker[['R1', 'R2', 'R3', 'R4', 'R5']].apply(lambda x: sum(x), axis=0)
print("Time using .apply(): {} sec".format(time.time() - apply_start_time))
```

```
Time using .apply(): 0.00490880012 seconds
```

```
start_time = time.time()
poker[['R1', 'R1', 'R3', 'R4', 'R5']].sum(axis=0)
print("Time using pandas: {} sec".format(time.time() - start_time))
```

```
Time using pandas: 0.00279092788 sec
```

```
Difference in speed: 160.310951649%
```

Let's do it!

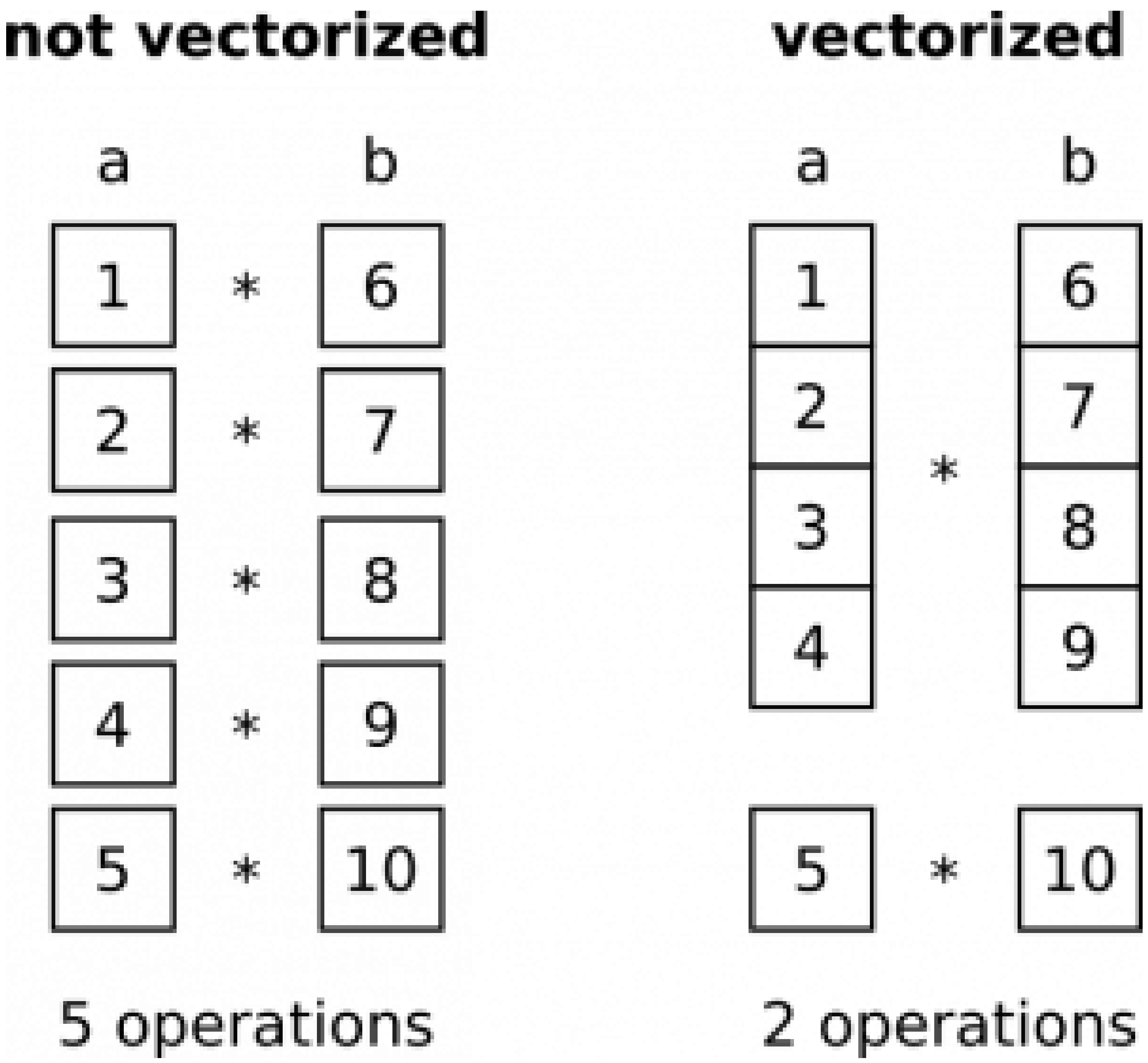
OPTIMIZING PYTHON CODE WITH PANDAS

Vectorization over Pandas series

OPTIMIZING PYTHON CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

DataFrames as arrays



How to perform pandas vectorization

```
start_time = time.time()
poker[['R1', 'R2', 'R3', 'R4', 'R5']].sum(axis=1)
print("Time using pandas vectorization: {} sec".format(time.time() - start_time))
```

```
Time using pandas vectorization: 0.0026819705 sec
```

```
poker[['R1', 'R2', 'R3', 'R4', 'R5']].sum(axis=1).head()
```

0	47
1	47
2	47
3	47
4	47
dtype: int64	--

Comparison to the previous methods

```
data_generator = data.iterrows()

start_time = time.time()
for index, value in data_generator:
        sum([value[1], value[3], value[5], value[7]])
print("Time using .iterrows(){} seconds" % (time.time() - start_time))
```

```
Results from the above operation calculated in 3.37918996 seconds
```

```
start_time = time.time()
data[['R1', 'R2', 'R3', 'R4', 'R5']].apply(lambda x: sum(x),axis=1)
print("Results from the above operation calculated in %s seconds" % (time.time() - start_time))
```

```
Results from the above operation calculated in 0.637711048 seconds
```

```
- Difference between vectorization and the `.iterows()` function: 111,800.75%
- Difference between vectorization and the `.apply()` function: 20,853%
```

Let's do it!

OPTIMIZING PYTHON CODE WITH PANDAS

Vectorization with NumPy arrays using .values()

OPTIMIZING PYTHON CODE WITH PANDAS

Leonidas Souliotis
PhD candidate

NumPy in pandas

```
df = pd.DataFrame({'Col1':[0, 1,
2, 3, 4, 5, 6]}, dtype=np.int8)
print(df)
```

```
|        | Col1 |
|--------|------|
| 0      | 0    |
| 1      | 1    |
| 2      | 2    |
| 3      | 3    |
| 4      | 4    |
| 5      | 5    |
| 6      | 6    |
```

```
nd = np.array(range(7))
print(nd)

[0 1 2 3 4 5 6]
```

How to perform NumPy vectorization

```
start_time = time.time()
poker[['R1', 'R2', 'R3', 'R4', 'R5']].values.sum(axis=1)
print("Time using NumPy vectorization: {} sec(time.time() - start_time))
```

```
Results from the above operation calculated in 0.00157618522644 seconds
```

```
start_time = time.time()
poker[['R1', 'R2', 'R3', 'R4', 'R5']].sum(axis=1)
print("Results from the above operation calculated in %s seconds" % (time.time() - start_time))
```

```
Results from the above operation calculated in 0.00268197059631 seconds
```

```
Difference in time: 39.0482%
```

Let's do it!

OPTIMIZING PYTHON CODE WITH PANDAS