The need for efficient coding I

OPTIMIZING PYTHON CODE WITH PANDAS

Leonidas Souliotis
PhD Researcher

How do we measure time?

`time.time()` : returns current time in seconds since 12:00am, January 1, 1970

```python
import time

# record time before execution
start_time = time.time()

# execute operation
result = 5 + 2

# record time after execution
end_time = time.time()

print("Result calculated in {} sec".format(end_time - start_time))
```

```
Result calculated in 9.48905944824e-05 sec
```

For loop vs List comprehension

- List comprehension:

```
list_comp_start_time = time.time()
result = [i*i for i in range(0,1000000)]
list_comp_end_time = time.time()
print("Time using the list_comprehension: {} sec".format(list_comp_end_time -
list_comp_start_time))
```

- For loop:

```
for_loop_start_time= time.time()
result=[]
for i in range(0,1000000):
  result.append(i*i)
for_loop_end_time= time.time()
print("Time using the for loop: {} sec".format(for_loop_end_time - for_loop_start_time))
```

For loop vs List comprehension II

```
Time using the list comprehension: 0.11042404174804688 sec
Time using the for loop: 0.2071230411529541 sec
```

```
list_comp_time = list_comp_end_time - list_comp_start_time
for_loop_time = for_loop_end_time - for_loop_start_time
print("Difference in time: {} %".format((for_loop_time - list_comp_time)/
list_comp_time*100))
```

```
Difference in time: 87.55527367398622 %
```

Where time matters I

Calculate $1 + 2 + ... + 1000000$.

- Adding numbers one by one:

```
def sum_brute_force(N):
    res = 0
    for i in range(1,N+1):
        res+=i
    return res
```

- Using $1 + 2 + ... + N = \frac{N \cdot (N+1)}{2}$

```
def sum_formula(N):
    return N*(N+1)/2
```

Where time matters II

- Using the formula:

```
# Using the formula
formula_start_time = time.time()
formula_result = formula(1000000)
formula_end_time = time.time()

print("Time using the formula: {}
sec".format(formula_end_time - formula_start_time
```

```
Using the formula: 0.000108957290649 sec
```

- Using brute force:

```
# Using brute force
bf_start_time = time.time()
bf_result = sum_brute_force(1000000)
bf_end_time = time.time()

print("Time using brute force: {}
sec".format(bf_end_time - start_time))
```

```
Time using brute force: 0.174870967865 sec
```

```
Difference in speed: 160,394.967179%
```

Let's do it!

OPTIMIZING PYTHON CODE WITH PANDAS

Locate rows: .iloc[] and .loc[]

OPTIMIZING PYTHON CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

The poker dataset

	S1	R1	S2	R2	S3	R3	S4	R4	S5	R5
1	♦	10	♣	Jack	♣	King	♠	4	♥	Ace
2	♦	Jack	♦	King	♦	10	♦	Queen	♦	Ace
3	♣	Queen	♣	Jack	♣	King	♣	10	♣	Ace

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

Sn: symbol of the n-th card

1 — Hearts, 2 — Diamonds, 3 — Clubs, 4 — Spades

Rn: rank of the n-th card

1 — Ace, 2-10, 11 — Jack, 12 — Queen, 13 — King

Locate targeted rows

`.loc[]` — index name locator

```
# Specify the range of rows to select
rows = range(0, 500)
# Time selecting rows using .loc[]
loc_start_time = time.time()
data.loc[rows]
loc_end_time = time.time()

print("Time using .loc[] : {} sec".format(
            loc_end_time - loc_start_time))
```

```
Time using .loc[]: 0.001951932 seconds
```

`.iloc[]` — index number locator

```
# Specify the range of rows to select
rows = range(0, 500)
# Time selecting rows using .iloc[]
iloc_start_time = time.time()
data.iloc[rows]
iloc_end_time = time.time()

print("Time using .iloc[]: {} sec".format(
            iloc_end_time - iloc_start_time)
```

```
Time using .iloc[] : 0.0007140636 sec
```

```
Difference in speed: 173.355592654%
```

Locate targeted columns

`.iloc[]` — index number locator

```
iloc_start_time = time.time()
data.iloc[:,:3]
iloc_end_time = time.time()
print("Time using .iloc[]: {} sec".format(
            iloc_end_time - iloc_start_time))
```

```
Time using .iloc[]: 0.00125193595886 sec
```

Locating columns by names

```
names_start_time = time.time()
data[['S1', 'R1', 'S2']]
names_end_time = time.time()
print("Time using selection by name: {} sec".for
            names_end_time - names_start_time))
```

```
Time using selection by name: 0.000964879989624
```

```
Difference in speed: 29.7504324188%
```

Let's do it!

OPTIMIZING PYTHON CODE WITH PANDAS

Select random rows

OPTIMIZING PYTHON CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

Sampling random rows using pandas

```
start_time = time.time()
poker.sample(100, axis=0)
print("Time using sample: {} sec".format(time.time() - start_time))
```

```
Time using sample: 0.000750064849854 sec
```

Sampling random rows using numpy

```
start_time = time.time()
poker.iloc[np.random.randint(low=0, high=poker.shape[0], size=100)]
print("Time using .iloc[]: {} sec".format(time.time() - start_time))
```

```
Time using .iloc[]: 0.00103211402893 sec
```

```
Difference in speed: 37.6033057849%
```

Sampling random columns

```
start_time = time.time()
poker.sample(3, axis=1)
print("Time using .sample(): {} sec".format(time.time() - start_time))
```

```
Time using .sample(): 0.000683069229126 sec
```

```
N = poker.shape[1]
start_time = time.time()
poker.iloc[:,np.random.randint(low=0, high=N, size=3)]
print("Time using .iloc[]: {} sec".format(time.time() - start_time))
```

```
ime using .iloc[]: 0.0010929107666 sec
```

```
Difference in speed: 59.9999999998%
```

Let's do it!

OPTIMIZING PYTHON CODE WITH PANDAS