The need for efficient coding I

WRITING EFFICIENT 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

```
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!

WRITING EFFICIENT CODE WITH PANDAS

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

WRITING EFFICIENT 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".format(
            names_end_time - names_start_time))
```

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

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

Let's do it!

Select random rows

WRITING EFFICIENT 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!

Replace scalar values using .replace()

WRITING EFFICIENT CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

The popular name dataset

Year of Birth	Gender	Ethnicity	Child's First Name	Count	Rank
2011	FEMALE	ASIAN AND PACIFIC ISLANDER	SOPHIA	119	1
2011	FEMALE	ASIAN AND PACIFIC ISLANDER	CHLOE	106	2

Replace values in pandas

```
start_time = time.time()
names['Gender'].loc[names.Gender=='MALE'] = 'BOY'
print("Replace values using .loc[]: {} sec".format(time.time() - start_time))
```

```
Results from the first method calculated in 0.0311849 seconds
```

Replace values using .replace()

```
start_time = time.time()
names['Gender'].replace('MALE', 'BOY', inplace=True)
print("Time using .replace(): {} sec".fomrat(time.time() - start_time))
```

```
Time using .replace(): 0.0016758441925 sec
```

```
Differerence in speed: 1,704.52411439%
```

Let's do it

Replace values using lists

WRITING EFFICIENT CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

Replace multiple values with one value

Year of Birth	Gender	Ethnicity	Child's First Name	Count	Rank
2011	FEMALE	WHITE NON HISP	HELENA	97	4

```
start_time = time.time()
names['Ethnicity'].loc[(names["Ethnicity"] == 'WHITE NON HISPANIC') |
(names["Ethnicity"] == 'WHITE NON HISP')] = 'WNH'
print("Results from the above operation calculated in %s seconds" %
 (time.time() - start_time))
```

```
Results from the second method calculated in 0.0276169776917 seconds
```

Replace multiple values using .replace() I

```
start_time = time.time()
names['Ethnicity'].replace(['WHITE NON HISPANIC','WHITE NON HISP'],
'WNH', inplace=True)
print("Time using .replace(): {} sec".format(time.time() - start_time))
```

```
Time using .replace():  0.00144791603088 sec
```

```
Difference in speed: 2160.68681809%
```

```python
names['Ethnicity'].replace(['WHITE NON HISP'], 'WHITE NON HISPANIC', inplace=True)
names['Ethnicity'].replace(['BLACK NON HISP'], 'BLACK NON HISPANIC', inplace=True)
```

```python
names['Ethnicity'].replace(['BLACK NON HISP','WHITE NON HISP'], ['BLACK NON HISPANIC',
'WHITE NON HISPANIC'], inplace=True)
```

Let's do it

Replace values using dictionaries

WRITING EFFICIENT CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

Replace single values with dictionaries

```
start_time = time.time()
names['Gender'].replace({'MALE':'BOY', 'FEMALE':'GIRL'},
inplace=True)
print("Time using .replace() with dictionary: {} sec".format(time.time() - start_time))
```

```
Time using .replace() with dictionary: 0.00197792053223 sec
```

```
start_time = time.time()
names['Gender'].replace('MALE', 'BOY', inplace=True)
names['Gender'].replace('FEMALE', 'GIRL', inplace=True)
print("Time using multiple .replace(): {} sec".format(time.time() - start_time))
```

```
Time using multiple .replace(): 0.00307083129883 sec
```

```
Difference in speed: 55.2555448407%
```

Replace multiple values using dictionaries

```
start_time = time.time()
names.replace({'Ethnicity': {'ASIAN AND PACI': 'ASIAN', 'ASIAN AND PACIFIC ISLANDER': 'ASIAN',
                             'BLACK NON HISPANIC': 'BLACK', 'BLACK NON HISP': 'BLACK',
                             'WHITE NON HISPANIC': 'WHITE', 'WHITE NON HISP': 'WHITE'}})

print("Time using .replace() with dictionary: {} sec".format (time.time() - start_time))
```

```
Time using .replace() with dictionary: 0.0028018 sec
```

Let's do it!

WRITING EFFICIENT CODE WITH PANDAS

Looping using the .iterrows() function

WRITING EFFICIENT 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!

WRITING EFFICIENT CODE WITH PANDAS

Looping using the .apply() function

WRITING EFFICIENT 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!

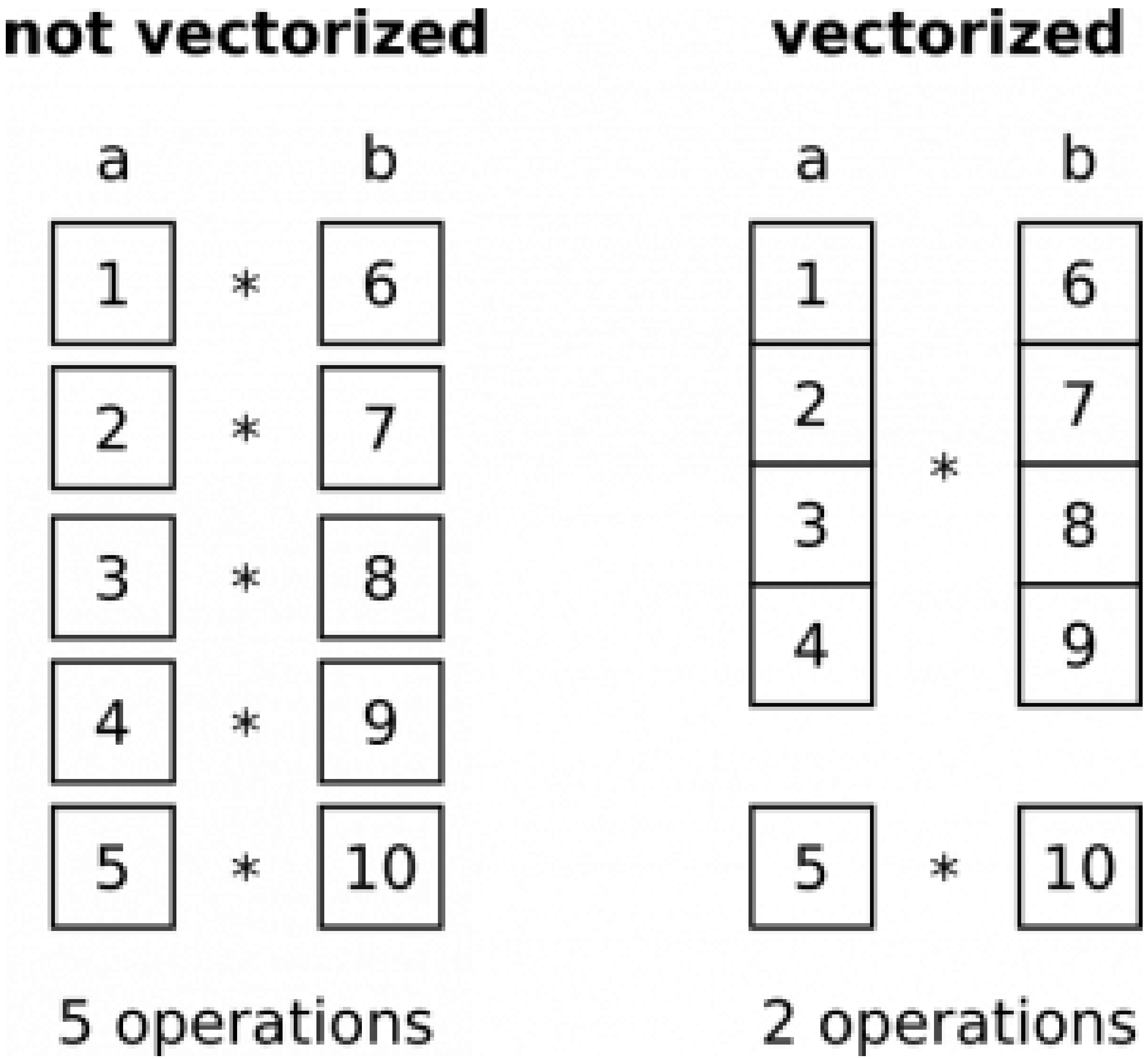
WRITING EFFICIENT CODE WITH PANDAS

Vectorization over Pandas series

WRITING EFFICIENT 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!

WRITING EFFICIENT CODE WITH PANDAS

Vectorization with NumPy arrays using .values()

WRITING EFFICIENT 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!

WRITING EFFICIENT CODE WITH PANDAS

Data transformation using .groupby().transform

WRITING EFFICIENT CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

The restaurant dataset

```
| total_bill    | tip    | sex        | smoker     | day    | time       |
|---------------|--------|------------|------------|--------|------------|
| 16.99         | 1.01   | Female     | No         | Sun    | "Dinner"   |
| 10.34         | 1.66   | Male       | No         | Sun    | "Dinner"   |
```

```
restaurant_grouped = restaurant.groupby('smoker')
```

```
print(restaurant_grouped.count())
```

```
|                   | total_bill |    tip |    sex  |  day    |   time |
|-------------------|------------|--------|---------|---------|--------|
| smoker            |            |        |         |         |        |
| No                | 151        | 151    | 151     | 151     | 151    |
| Yes               | 93         | 93     | 93      | 93      | 93     |
```

Data transformation

```
zscore = lambda x: (x - x.mean()  ) / x.std()
```

```
restaurant_grouped = restaurant.groupby('time')
restaurant_transformed = restaurant_grouped.transform(zscore)
```

```
restaurant_transformed.head()
```

```
   total_bill       tip      size
0   -0.416446 -1.457045 -0.692873
1   -1.143855 -1.004475  0.405737
2    0.023282  0.276645  0.405737
3    0.315339  0.144355 -0.692873
```

Comparison with native methods

```
restaurant.groupby('sex').transform(zscore)

mean_female = restaurant.groupby('sex').mean()['total_bill']['Female']
mean_male = restaurant.groupby('sex').mean()['total_bill']['Male']
std_female = restaurant.groupby('sex').std()['total_bill']['Female']
std_male = restaurant.groupby('sex').std()['total_bill']['Male']

for i in range(len(restaurant)):
    if restaurant.iloc[i][2] == 'Female':
        restaurant.iloc[i][0] = (restaurant.iloc[i][0] - mean_female)/std_female
    else:
        restaurant.iloc[i][0] = (restaurant.iloc[i][0] - mean_male)/std_male
```

```
Time using .groupby(): 0.016291141 seconds
Time using native Python: 3.937326908 seconds

Difference in time: 24,068.5145%
```

Let's practice!

WRITING EFFICIENT CODE WITH PANDAS

Missing value imputation using transform()

WRITING EFFICIENT CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

Counting missing values

```
prior_counts = restaurant.groupby('time')
['total_bill'].count()
```

```
missing_counts = restaurant_nan.groupby('time')
['total_bill'].count()
print(prior_counts - missing_counts)
```

```
time
Dinner    32
Lunch     13
Name: total_bill, dtype: int64
```

Missing value imputation

```
missing_trans = lambda x: x.fillna(x.mean())
```

```
restaurant_nan_grouped = restaurant_nan.groupby('time')['total_bill']
restaurant_nan_grouped.transform(missing_trans)
```

```
Time using .transform(): 0.00368881225586 sec
```

```
0     20.676573
1     10.340000
2     21.010000
3     23.680000
4     24.590000
5     25.290000
6     20.676573
Name: total_bill, dtype: float64
```

Comparison with native methods

```
start_time = time.time()
mean_din = restaurant_nan.loc[restaurant_nan.time ==
'Dinner']['total_bill'].mean()
mean_lun = restaurant_nan.loc[restaurant_nan.time ==
'Lunch']['total_bill'].mean()

for row in range(len(restaurant_nan)):
    if restaurant_nan.iloc[row]['time'] == 'Dinner':
        restaurant_nan.loc[row, 'total_time'] = mean_din
    else:
        restaurant_nan.loc[row, 'total_time'] = mean_lun
print("Results from the above operation calculated in %s seconds" % (time.time() - start_time))
```

```
Time using native Python: 0.172566890717 sec
```

```
Difference in time: 4,578.115%
```

Let's do it!

WRITING EFFICIENT CODE WITH PANDAS

Data filtration using the filter() function

WRITING EFFICIENT CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

Purpose of filter()

Limit results based on an aggregate feature

- Number of missing values
- Mean of a specific feature
- Number of occurrences of the group

Filter using groupby().filter()

```
restaurant_grouped = restaurant.groupby('day')
filter_trans = lambda x : x['total_bill'].mean() > 20
restaurant_filtered = restaurant_grouped.filter(filter_trans)
```

```
Time using .filter() 0.00414085388184 sec
```

```
print(restaurant_filtered['tip'].mean())
```

```
3.11527607362
```

```
print(restaurant['tip'].mean())
```

```
2.9982786885245902
```

Comparison with native methods

```
t=[restaurant.loc[df['day'] == i]['tip'] for i in restaurant['day'].unique()
    if restaurant.loc[df['day'] == i]['total_bill'].mean()>20]
restaurant_filtered = t[0]
for j in t[1:]:
    restaurant_filtered=restaurant_filtered.append(j,ignore_index=True)
```

```
Time using native Python: 0.00663900375366 sec
```

```
print(restaurant_filtered.mean())
```

```
3.11527607362
```

```
Difference in time: 60.329341317157024%
```

Let's do it!

WRITING EFFICIENT CODE WITH PANDAS

Congratulations!

WRITING EFFICIENT CODE WITH PANDAS

Leonidas Souliotis
PhD Candidate

What you have learned

- Why and how to time operations
- Select targeted rows and columns efficiently
- Select random rows and columns efficiently
- Replace values of a DataFrame efficiently using replace()
  - Replace multiple values using lists
  - Replace multiple values using dictionaries

What you have learned

- Iterate on a DataFrame using the .iterrows() function
- Iterate on a DataFrame using the .apply() function
- Iterate on a DataFrame using pandas optimization
- Iterate on a DataFrame using numpy optimization
- Comparison of the groupby() function compared to native python code
  - When transforming the data group-wise
  - When imputing missing values group-wise
  - When filtering groups with specific characteristics

Congratulations!

WRITING EFFICIENT CODE WITH PANDAS