Intro to pandas DataFrame iteration

WRITING EFFICIENT PYTHON CODE

Logan Thomas
Scientific Software Technical Trainer, Enthought

datacamp

pandas recap

- See pandas overview in **Intermediate Python**
- Library used for data analysis
- Main data structure is the DataFrame
  - Tabular data with labeled rows and columns
  - Built on top of the NumPy array structure
- Chapter Objective:
  - Best practice for iterating over a pandas DataFrame

Baseball stats

```
import pandas as pd


baseball_df = pd.read_csv('baseball_stats.csv')
print(baseball_df.head())
```

```
  Team League  Year   RS   RA   W    G  Playoffs
0  ARI     NL  2012  734  688  81  162         0
1  ATL     NL  2012  700  600  94  162         1
2  BAL     AL  2012  712  705  93  162         1
3  BOS     AL  2012  734  806  69  162         0
4  CHC     NL  2012  613  759  61  162         0
```

Baseball stats

```
  Team
0  ARI
1  ATL
2  BAL
3  BOS
4  CHC
```

Arizona Diamondbacks (ARI)

Atlanta Braves (ATL)

Baltimore Orioles (BAL)

Boston Red Sox (BOS)

Chicago Cubs (CHC)

Baseball stats

	Team	League	Year	RS	RA	W	G	Playoffs
0	ARI	NL	2012	734	688	81	162	0
1	ATL	NL	2012	700	600	94	162	1
2	BAL	AL	2012	712	705	93	162	1
3	BOS	AL	2012	734	806	69	162	0
4	CHC	NL	2012	613	759	61	162	0

Calculating win percentage

```
import numpy as np

def calc_win_perc(wins, games_played):

    win_perc = wins / games_played

    return np.round(win_perc,2)
```

```
win_perc = calc_win_perc(50, 100)
print(win_perc)
```

```
0.5
```

Adding win percentage to DataFrame

```
win_perc_list = []
for i in range(len(baseball_df)):
    row = baseball_df.iloc[i]
    wins = row['W']
    games_played = row['G']
    win_perc = calc_win_perc(wins, games_played)
    win_perc_list.append(win_perc)
baseball_df['WP'] = win_perc_list
```

Adding win percentage to DataFrame

```
print(baseball_df.head())
```

```
  Team League  Year   RS   RA   W    G  Playoffs    WP
0  ARI     NL  2012  734  688  81  162         0  0.50
1  ATL     NL  2012  700  600  94  162         1  0.58
2  BAL     AL  2012  712  705  93  162         1  0.57
3  BOS     AL  2012  734  806  69  162         0  0.43
4  CHC     NL  2012  613  759  61  162         0  0.38
```

Iterating with .iloc

```
%%timeit
win_perc_list = []

for i in range(len(baseball_df)):
    row = baseball_df.iloc[i]

    wins = row['W']
    games_played = row['G']

    win_perc = calc_win_perc(wins, games_played)
    win_perc_list.append(win_perc)

baseball_df['WP'] = win_perc_list
```

```
183 ms ± 1.73 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Iterating with .iterrows()

```
win_perc_list = []

for i,row in baseball_df.iterrows():
    wins = row['W']
    games_played = row['G']

    win_perc = calc_win_perc(wins, games_played)

    win_perc_list.append(win_perc)

baseball_df['WP'] = win_perc_list
```

Iterating with .iterrows()

```
%%timeit
win_perc_list = []

for i,row in baseball_df.iterrows():

    wins = row['W']
    games_played = row['G']

    win_perc = calc_win_perc(wins, games_played)
    win_perc_list.append(win_perc)

baseball_df['WP'] = win_perc_list
```

```
95.3 ms ± 3.57 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Practice DataFrame iterating with .iterrows()

WRITING EFFICIENT PYTHON CODE

Another iterator method: .itertuples()

WRITING EFFICIENT PYTHON CODE

Logan Thomas
Scientific Software Technical Trainer, Enthought

datacamp

Team wins data

```
print(team_wins_df)
```

```
     Team  Year    W
0     ARI  2012   81
1     ATL  2012   94
2     BAL  2012   93
3     BOS  2012   69
4     CHC  2012   61
...
```

```
for row_tuple in team_wins_df.iterrows():
    print(row_tuple)
    print(type(row_tuple[1]))
```

```
(0, Team     ARI
Year    2012
W         81
Name: 0, dtype: object)
<class 'pandas.core.series.Series'>

(1, Team     ATL
Year    2012
W         94
Name: 1, dtype: object)
<class 'pandas.core.series.Series'>
...
```

Iterating with .itertuples()

```
for row_namedtuple in team_wins_df.itertuples():
    print(row_namedtuple)
```

```
Pandas(Index=0, Team='ARI', Year=2012, W=81)
Pandas(Index=1, Team='ATL', Year=2012, W=94)
...
```

```
print(row_namedtuple.Index)
```

```
1
```

```
print(row_namedtuple.Team)
```

```
ATL
```

Comparing methods

```
%%timeit
for row_tuple in team_wins_df.iterrows():
    print(row_tuple)
```

```
527 ms ± 41.1 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%%timeit
for row_namedtuple in team_wins_df.itertuples():
    print(row_namedtuple)
```

```
7.48 ms ± 243 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

```
for row_tuple in team_wins_df.iterrows():
    print(row_tuple[1]['Team'])
```

```
ARI
ATL
...
```

```
for row_namedtuple in team_wins_df.itertuples():
    print(row_namedtuple['Team'])
```

```
TypeError: tuple indices must be integers or slices, not str
```

```
for row_namedtuple in team_wins_df.itertuples():
    print(row_namedtuple.Team)
```

```
ARI
ATL
...
```

Let's keep iterating!

WRITING EFFICIENT PYTHON CODE

pandas alternative to looping

WRITING EFFICIENT PYTHON CODE

Logan Thomas
Scientific Software Technical Trainer, Enthought

```
print(baseball_df.head())
```

```
  Team League  Year   RS   RA   W    G  Playoffs
0  ARI     NL  2012  734  688  81  162         0
1  ATL     NL  2012  700  600  94  162         1
2  BAL     AL  2012  712  705  93  162         1
3  BOS     AL  2012  734  806  69  162         0
4  CHC     NL  2012  613  759  61  162         0
```

```
def calc_run_diff(runs_scored, runs_allowed):

    run_diff = runs_scored - runs_allowed

    return run_diff
```

Run differentials with a loop

```
run_diffs_iterrows = []

for i,row in baseball_df.iterrows():
    run_diff = calc_run_diff(row['RS'], row['RA'])
    run_diffs_iterrows.append(run_diff)

baseball_df['RD'] = run_diffs_iterrows
print(baseball_df)
```

```
     Team League  Year    RS   RA    W    G  Playoffs   RD
0     ARI     NL  2012   734  688   81  162         0   46
1     ATL     NL  2012   700  600   94  162         1  100
2     BAL     AL  2012   712  705   93  162         1    7
...
```

pandas .apply() method

- Takes a function and applies it to a DataFrame
  - Must specify an axis to apply (`0` for columns; `1` for rows)
- Can be used with anonymous functions (`lambda` functions)
- Example:

```
baseball_df.apply(
    lambda row: calc_run_diff(row['RS'], row['RA']),
    axis=1
)
```

Run differentials with .apply()

```
run_diffs_apply = baseball_df.apply(
          lambda row: calc_run_diff(row['RS'], row['RA']),
          axis=1)
baseball_df['RD'] = run_diffs_apply
print(baseball_df)
```

```
     Team League  Year    RS    RA     W     G  Playoffs    RD
0     ARI     NL  2012   734   688    81   162          0    46
1     ATL     NL  2012   700   600    94   162          1   100
2     BAL     AL  2012   712   705    93   162          1     7
...
```

Comparing approaches

```
%%timeit
run_diffs_iterrows = []

for i,row in baseball_df.iterrows():
    run_diff = calc_run_diff(row['RS'], row['RA'])
    run_diffs_iterrows.append(run_diff)

baseball_df['RD'] = run_diffs_iterrows
```

```
86.8 ms ± 3 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Comparing approaches

```
%%timeit
run_diffs_apply = baseball_df.apply(
          lambda row: calc_run_diff(row['RS'], row['RA']),
          axis=1)

baseball_df['RD'] = run_diffs_apply
```

```
30.1 ms ± 1.75 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

Let's practice using pandas .apply() method!

WRITING EFFICIENT PYTHON CODE

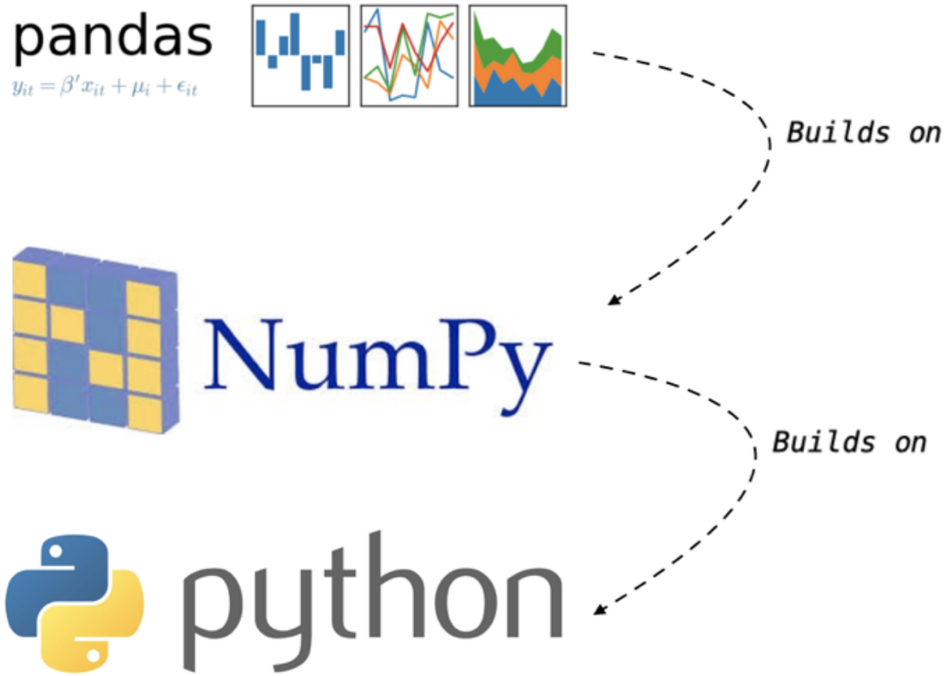
Optimal pandas iterating

WRITING EFFICIENT PYTHON CODE

Logan Thomas
Scientific Software Technical Trainer, Enthought

pandas internals

- Eliminating loops applies to using pandas as well
- pandas is built on NumPy
  - Take advantage of NumPy array efficiencies



```
print(baseball_df)
```

```
  Team League  Year   RS   RA   W    G  Playoffs
0  ARI     NL  2012  734  688  81  162         0
1  ATL     NL  2012  700  600  94  162         1
2  BAL     AL  2012  712  705  93  162         1
...
```

```
wins_np = baseball_df['W'].values
print(type(wins_np))
```

```
<class 'numpy.ndarray'>
```

```
print(wins_np)
```

```
[ 81  94  93 ...]
```

Power of vectorization

- Broadcasting (vectorizing) is extremely efficient!

```
baseball_df['RS'].values - baseball_df['RA'].values
```

```
array([  46,  100,    7, ...,  188,  110, -117])
```

Run differentials with arrays

```
run_diffs_np = baseball_df['RS'].values - baseball_df['RA'].values
baseball_df['RD'] = run_diffs_np
print(baseball_df)
```

```
     Team League  Year   RS   RA    W    G  Playoffs   RD
0     ARI     NL  2012  734  688   81  162         0   46
1     ATL     NL  2012  700  600   94  162         1  100
2     BAL     AL  2012  712  705   93  162         1    7
3     BOS     AL  2012  734  806   69  162         0  -72
4     CHC     NL  2012  613  759   61  162         0 -146
...
```

Comparing approaches

```
%%timeit
run_diffs_np = baseball_df['RS'].values - baseball_df['RA'].values


baseball_df['RD'] = run_diffs_np
```

```
124 µs ± 1.47 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

Let's put our skills into practice!

WRITING EFFICIENT PYTHON CODE

Congratulations!

WRITING EFFICIENT PYTHON CODE

Logan Thomas
Scientific Software Technical Trainer, Enthought

What you have learned

- The definition of **efficient** and **Pythonic** code
- How to use Python's powerful built-in library
- The advantages of NumPy arrays
- Some handy magic commands to profile code
- How to deploy efficient solutions with `zip()`, `itertools`, `collections`, and set theory
- The cost of looping and how to eliminate loops
- Best practices for iterating with pandas DataFrames

Well done!

WRITING EFFICIENT PYTHON CODE