Programming in Python[1]

Mattia Monga

Dip. di Informatica
Università degli Studi di Milano, Italia
mattia.monga@unimi.it

Academic year 2022/23, I semester

Lecture XVIII: More pandas

Group by

Data can be grouped with `groupby`, then *summary* function (`sum`, `mean`, ...) can be applied to each group at the same time.

```
iris = pd.read_csv('https://tinyurl.com/iris-data')
```

```
iris.groupby('variety').mean()
```

Groups are special lazy types which generate data only when needed for the summary operation.

Iterators

Object can be iterable. Python defines the iterator protocol as:

- `iterator.__iter__()` Return the iterator object itself. This is required to allow both containers and iterators to be used with the for and in statements.
- `iterator.__next__()` Return the next item from the container. If there are no further items, raise the StopIteration exception.

Notable iterators

Built-in lists, tuples, ranges, sets, dicts are iterators.

- Numpy arrays
- Pandas Series and DataFrames

Generators


```
def mygenerator() -> int:
    for i in [1, 6, 70, 2]:
        yield i
    print('Ended')  # Just to see when it reaches this
    ↪  point

g = mygenerator()

print(g)  # not useful
print(next(g))
print(next(g))
print(next(g))
print(next(g))
print(next(g))      # Exception
```

Pandas DataFrame

Be careful: the default iteration is on column names (similar to dicts, which iterate on keys).

- `iterrows()`: Iterate over the rows of a DataFrame as `(index, Series)` pairs. This converts the rows to Series objects, which can change the dtypes and has some performance implications.
- `itertuples()`: Iterate over the rows of a DataFrame as namedtuples of the values. This is a lot faster than `iterrows()`, and is in most cases preferable to use to iterate over the values of a DataFrame.

Iterating is slow: whenever possibile try to use vectorized operation or function application.

Pandas function application


```python
# apply the function to each column
df.apply(lambda col: col.mean() + 3)

# apply the function to each row
df.apply(lambda row: row + 3, axis=1)
```

Pandas query


```
df[df['A A'] > 3]

# equivalent to this (backticks because of the space)
df.query('`A A` > 3')

# query can also refer to the index
df.query('index >= 15')

# same as
df[15:]
```