Programming in Python[1]

Mattia Monga

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

Academic year 2025/26, I semester

Lecture XVIII: Tabular data

Tabular data

Data are often given/collected as tables: matrices with rows for individual records and columns for the fields of the records. This is especially common in statistics, R has a built-in type for this: the dataframe.

`pandas`

`pandas` (Python for data analysis) brings the `DataFrame` type to Python. It is based on `numpy`.

- `Series`: a one-dimensional labeled array capable of holding any data type (integers, strings, floating point numbers, Python objects, etc.). The axis labels are collectively referred to as the index.
- `DataFrame`: a 2-dimensional labeled data structure with columns of potentially different types. You can think of it like a spreadsheet, or a `dict` of `Series` objects.

Series


```
import pandas as pd
s = pd.Series(np.random.randn(5), index=["a", "b", "c",
↪  "d", "e"])
```

s is a numpy array of floats, each one has a label.

```
d = {"b": 1, "a": 0, "c": 2}

s = pd.Series(d)
```

The ordering depends on Python and pandas version... The current ones takes the insertion order, but you can provide explicitly the index.

```
d = {"b": 1, "a": 0, "c": 2}

s = pd.Series(d, index=['a', 'b', 'c'])
```


Series

A Series is convenient because it is a ndarray (and can be vectorized) but also a dict.

Dataframes


```
d = { "one": pd.Series([1.0, 2.0, 3.0], index=["a",
↪  "b", "c"]),
      "two": pd.Series([1.0, 2.0, 3.0, 4.0],
      ↪  index=["a", "b", "c", "d"]),
    }

df = pd.DataFrame(d)
```

A DataFrame has an index and a columns attribute.
There are many ways of creating DataFrames, see the docs.

From csv or spreadsheets

A famous example: Fisher's Iris flowers dataset.
150 records, "sepal length","sepal width","petal length","petal width","class"

```
iris = pd.read_csv('iris.csv')
# with a url
iris = pd.read_csv('https://tinyurl.com/iris-data')
```

Two ways of indexing

- `.loc[]` "label based"
- `.iloc[]` "position based"

For both you can use: a single value, a list of values, a boolean array. Two notable things:

1. If you use a slice notation with `.loc` (`'a':'f'`) the last value is included! (different from plain python and from `.iloc`)
2. Can be also a callable function with one argument (the calling Series or DataFrame) and that returns valid output for indexing (one of the above)

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.