Week 2: Data Representations

Dr Giuseppe Brandi

Northeastern University London

Outline

- Logical *vs.* machine data representations in NumPy
- Array slicing
- *n*-ary relations with Pandas

Why NumPy

Python has made data science accessible to many due to ease of programming. But as data grows, **performance matters.**

So, what is NumPy?

- A high level language to describe data naturally
- A low level, performant machine representation

Machine representations

- Arrays are stored in 1-D contiguous memory. For example,
- Given an $N \times M$ array A, $A_{ij} = A_{i \times N + j}$
- Contrary to Python `lists`, loops are more efficient
- Arrays can be stored in *row-major* or *column-major* order

Machine representations

Data representation **Machine representation**

$A =$

1	0	-2
3	5	2
4	1	6

1	0	-2	3	5	2	4	1	6
0	1	2	3	4	5	6	7	8

Row major (default)

$B =$

1	0	0
7	1	-2
4	3	1

1	7	4	0	1	3	0	-2	1
0	1	2	3	4	5	6	7	8

Column major

- Consider $A \times 2$
- Consider $A \times B$

Machine representations

Separating data intent from machine representations enables:

- Use of cache locality
- Use of architecture-specific vectorised instructions
- Use of optimised functions written in C or Fortran[1]

[1] See BLAS.

An array slice is a "view", not a copy

Unlike `lists`, NumPy array slices are "views" to the same memory location.

Example

```
1 a = np.array([1, 2, 3, 4])
2 b = a[:2]  # [1, 2]
3 b[0] = 0
4 print(a, 'and', b)
```

```
[0, 2, 3, 4] and [0, 2]
```

Outline

- Logical *vs.* machine data representations in NumPy
- Array slicing
- *n*-ary relations with Pandas

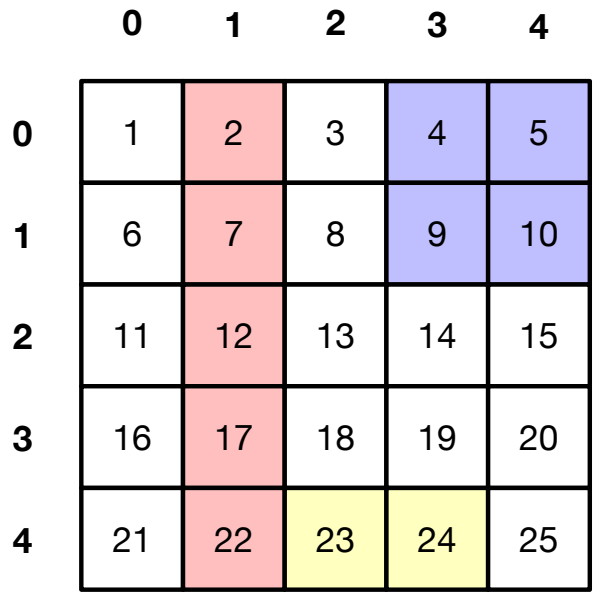
Array slicing

Much like slicing `list` objects, we can extract a sequence from an *n*-dimensional array, `a`:

```
a[lower:upper:stride, ...]
```

- `lower` is included
- `upper` is excluded
- Omitted indices, e.g. `[:2]` or `[1:]`, default to `0` or `len`
- Negative indices work also (-1 being the last)

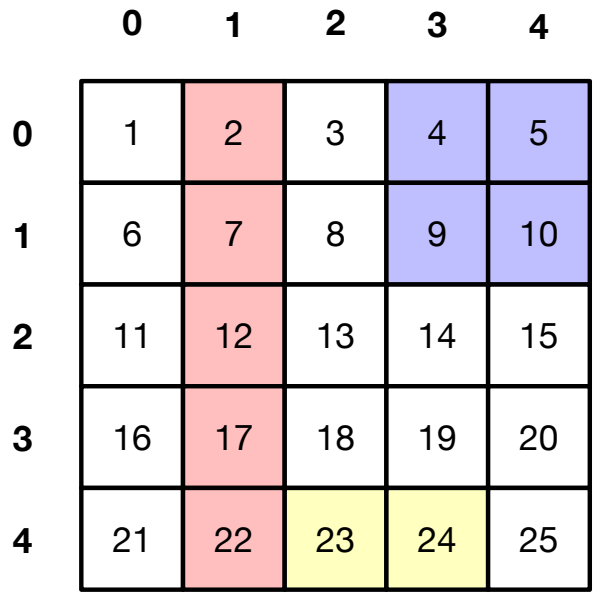
Array slicing



	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

```
1 a = np.arange(1, 26).reshape(5, 5)
2 salmon =
3 purple =
4 yellow =
```

Array slicing



	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

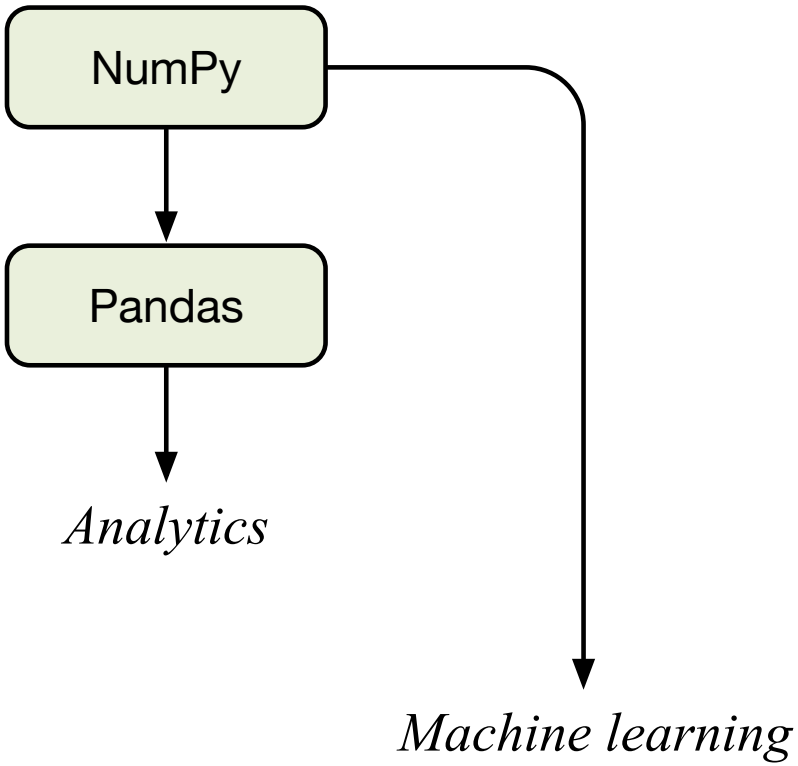
```
1 a = np.arange(1, 26).reshape(5, 5)
2 salmon = a[:, 1]
3 purple = a[:2, 3:]
4 yellow = a[-1, 2:4]
```

Outline

- Logical *vs.* machine data representations in NumPy
- Array slicing
- *n*-ary relations with Pandas

Why Pandas

From *n*-dimensional arrays to ***n*-ary relations**



Properties to keep in mind

- All rows are distinct
- The ordering of rows is insignificant
- The ordering of columns is significant, and consequently the labelling of columns matters
- One or more columns *uniquely identify* each row

Properties to keep in mind

Pandas is just a column store. We are responsible for the **integrity** of our data and, consequently, of our results.

A `DataFrame` can have duplicate rows

```
1 df = pd.DataFrame([[3, 4, 'A'], [3, 4, 'A']])
2 df.drop_duplicates(inplace=True)
```

```
   0  1  2
0  3  4  A
```

Properties to keep in mind

Pandas is just a column store. We are responsible for the **integrity** of our data and, consequently, of our results.

A `DataFrame` can have duplicate column names

```
1 df = pd.DataFrame([[3, 4, 'A'], [1, 2, 'B']])
2 df.columns = ['X', 'X']
3 df['X']
```

```
   X  X
0  3  1
1  4  2
2  A  B
```

Properties to keep in mind

Pandas is just a column store. We are responsible for the **integrity** of our data and, consequently, of our results.

Rows in a `DataFrame` can have the same index

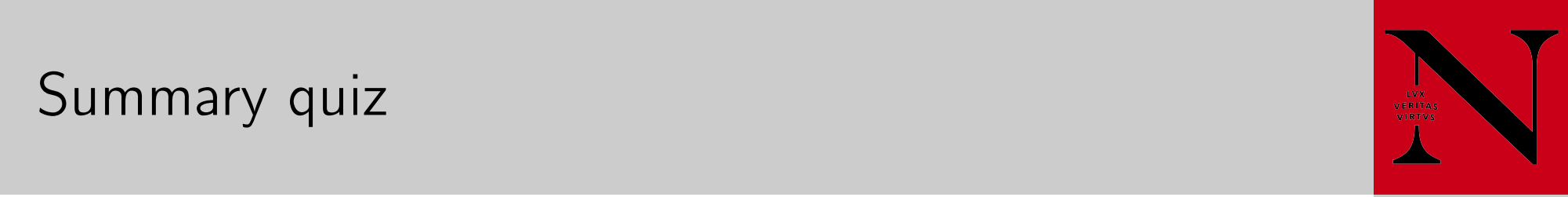
```
1 df = pd.DataFrame({'X': [3, 3], 'Y': [2, 1]})
2 df = df.set_index('X', verify_integrity=True)
```

```
ValueError: Index has duplicate keys
```

Summary

- `numpy` and `pandas` try to balance ease of programmability and performance
- Slicing creates *data views*, which are at the heart of many transformations
- `pandas` provide a relational view of data
- The integrity of relations is in our hands

Summary quiz

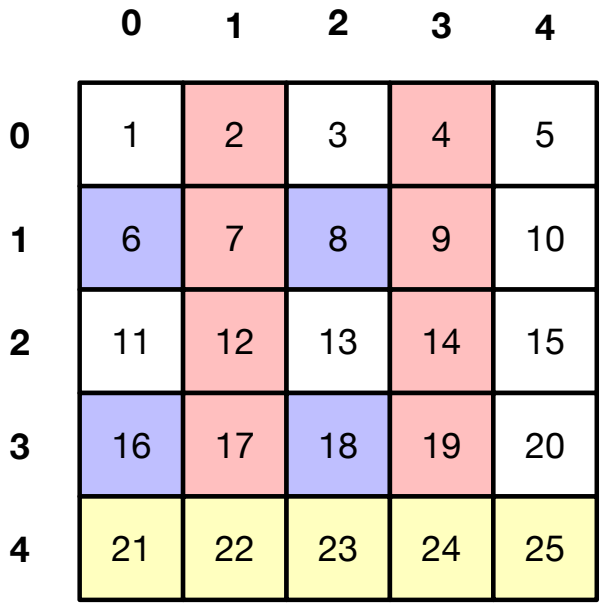


1. Find the slices of the array that match the coloured areas.

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

```
a = np.arange(1, 26).reshape(5, 5)
salmon = purple = yellow = None
```

Array slicing



	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

```
1 a = np.arange(1, 26).reshape(5, 5)
2 salmon = a[:4,1::2]
3 purple = a[1::2, 0:4:2]
4 yellow = a[-1,:]
```