Lecture VII - NumPy and Pandas for Scientific Computing

Programming with Python

Dr. Tobias Vlćek

Quick Recap of the last Lecture

NumPy Module

What is NumPy?

- **NumPy** is a package for scientific computing in Python
- Provides support for large, multi-dimensional arrays and matrices
- Wide range of mathematical functions to operate on these arrays
- Python lists can be slow - Numpy arrays are much faster

. . .

ℹ Note

The name of the package comes from Numerical Python.

Why is NumPy so fast?

- NumPy arrays are stored in a contiguous block of memory
- This allows for efficient memory access patterns
- Operations are implemented in the languages `C` and `C++`

How to get started

1. Install NumPy using `pip install numpy`
2. In Thonny, use `Tools -> Manage Packages...` to install NumPy
3. Import NumPy in a script using `import numpy as np`
4. You are ready to go!

. . .

```python
import numpy as np
x = np.array([1, 2, 3, 4, 5])
type(x)
```

```
numpy.ndarray
```

. . .

ℹ Note

You don't have to use `as np`. But it is a common practice to do so.

Creating Arrays

- The backbone of Numpy is the so called `ndarray`
- Can be initialized from different data structures:

```
import numpy as np
array_from_list = np.array([1, 1, 1, 1])
print(array_from_list)
```

```
[1 1 1 1]
```

```
import numpy as np
array_from_tuple = np.array((2, 2, 2, 2))
print(array_from_tuple)
```

```
[2 2 2 2]
```

Hetergenous Data Types

- It is possible to store different data types in a `ndarray`

```
import numpy as np
array_different_types = np.array(["s", 2, 2.0, "i"])
print(array_different_types)
```

```
['s' '2' '2.0' 'i']
```

. . .

> Note
>
> Not recommended, as it can lead to performance issues. I possible, **keep them homogenous**.

Creating Prefilled Arrays

Often used to improve performance by **allocating memory upfront**

- `np.zeros(shape)`: to create an array of zeros
- `np.ones(shape)`: to create an array of ones
- `np.random.rand(shape)`: to create an array of random values
- `np.arange(start, stop, step)`: evenly spaced values
- `np.linspace(start, stop, num)`: evenly spaced values

. . .

> Note
>
> The shape refers to the size of the array. It can have one or multiple dimensions.

Dimensions

- The shape is specified as tuple in these arrays
- `(2)` or `2` creates a 1-dimensional array (vetor)

- `(2,2)` creates a 2-dimensional array (matrix)
- `(2,2,2)` creates a 3-dimensional array (3rd order tensor)
- `(2,2,2,2)` creates a 4-dimensional array (4th order tensor)
- …

Arrays in Action

Task: Complete the following task

```
# Create a 3-dimensional tensor with filled with ones
# You can choose the shape of the tensor, but it should have 200 elements
# Sum over all values of the tensor
# Print the shape of the tensor using the method shape()
# Print the dtype of the tensor using the method dtype()
# Print the size of the tensor using the method size()
```

Indexing and Slicing

- Accessing and slicing `ndarray` elements works as before
- In higher dimensions we can access elements using multiple indices

. . .

Question: What do you expect will be printed?

```
import numpy as np
x = np.random.randint(0, 10, size=(3, 3))
print(x)
print("---")
print(x[0:2,0:2])
```

```
[[2 3 2]
 [3 4 0]
 [2 8 8]]
---
[[2 3]
 [3 4]]
```

Data Types

- Numpy provides data types as characters
- `i`: integer
- `b`: boolean
- `f`: float
- `S`: string
- `U`: unicode
- The type can be checked by calling the `.dtype` attribute

```
string_array = np.array(["Hello", "World"])
string_array.dtype
```

```
dtype('<U5')
```

Enforcing Data Types

- We can also **provide** the type when creating arrays

. . .

```
x = np.array([1, 2, 3, 4, 5],  dtype = 'f')
print(x.dtype)
```

```
float32
```

. . .

- Or we can **change** them for existing arrays

```
x = np.array([1, 2, 3, 4, 5],  dtype = 'f')
print(x.astype('i').dtype)
```

```
int32
```

. . .

> ℹ Note
>
> Note, how the types are specified as `int32` and `float32`.

Joining Arrays

- You can use `concatenate` two **join arrays**
- With `axis` you can specify the dimension
- Even easier in 2-dimensions is `hstack()` and `vstack()`

. . .

Question: What do you expect will be printed?

```
import numpy as np
ones = np.array((1,1,1,1))
twos = np.array((1,1,1,1)) *2
print(np.vstack((ones,twos)))
print(np.hstack((ones,twos)))
```

```
[[1 1 1 1]
 [2 2 2 2]]
[1 1 1 1 2 2 2 2]
```

Common Methods

- `sort()`: sort the array from low to high
- `reshape()`: reshape the array into a new shape
- `flatten()`: flatten the array into a 1D array
- `squeeze()`: squeeze the array to remove 1D entries
- `transpose()`: transpose the array

. . .

💡 Tip

Try experiment with these methods, as they can make your work later much easier.

Iterating over Arrays

- Naturally, we can also loop over a `ndarray`
-

Ufuncs

- To increase the speed even further, we can use

Filter

-

Pandas Module

What is Pandas?

- Pandas is a data manipulation and analysis library
- It provides data structures like **DataFrames and Series**
- It also provides tools for data cleaning, analysis, and visualization

How to install Pandas

Creating DataFrames

- Use the `pd.DataFrame(data, index, columns)` function to create a DataFrame from a dictionary
- Use the `pd.DataFrame(data, index, columns)` function to create a DataFrame from a dictionary
- Use the `pd.DataFrame(data, index, columns)` function to create a DataFrame from a dictionary

Basic Operations

- Use the `df.head()` method to display the first few rows of a DataFrame
- Use the `df.tail()` method to display the last few rows of a DataFrame
- Use the `df.info()` method to display information about a DataFrame
- Use the `df.describe()` method to display summary statistics about a DataFrame
- Use the `df.columns` attribute to access the column names of a DataFrame
- Use the `df.index` attribute to access the index of a DataFrame
- Use the `df.values` attribute to access the values of a DataFrame

Subsetting DataFrames

- Use the `df.loc[row_indexer, column_indexer]` method to access a specific element of a DataFrame
- Use the `df.iloc[row_indexer, column_indexer]` method to access a specific element of a DataFrame

Filtering DataFrames

- Use the `df[df['column'] > value]` method to filter a DataFrame
- Use the `df[df['column'].isin(values)]` method to filter a DataFrame

Grouping DataFrames

- Use the `df.groupby('column').sum()` method to group a DataFrame and calculate the sum of a column
- Use the `df.groupby('column').mean()` method to group a DataFrame and calculate the mean of a column
- Use the `df.groupby('column').count()` method to group a DataFrame and count the number of elements in a column
- Use the `df.groupby('column').size()` method to group a DataFrame and count the number of elements in a column

Excel Files

- Excel files can be read using the `pd.read_excel(file_path)` function
- Excel files can be written using the `df.to_excel(file_path)` method