YData: Introduction to Data Science

Class 05: Array computation

Overview

Brief review of what we have covered so far

NumPy arrays

Numerical array computations

Higher dimensional numerical arrays

Image manipulation

Very quick review of topics we have covered…

Expressions and type

- my_num = 2 * 3
- my_string = 'cat' + ' ' + 'hat'

List and dictionaries

- my_list = [1, 2, 3 , 4, 5, 'six'] # create a list
- my_list2 = my_list[0:3] # get the first 3 elements
- my_dict = { 'a': 7, 'b': 20} # create a dictionary

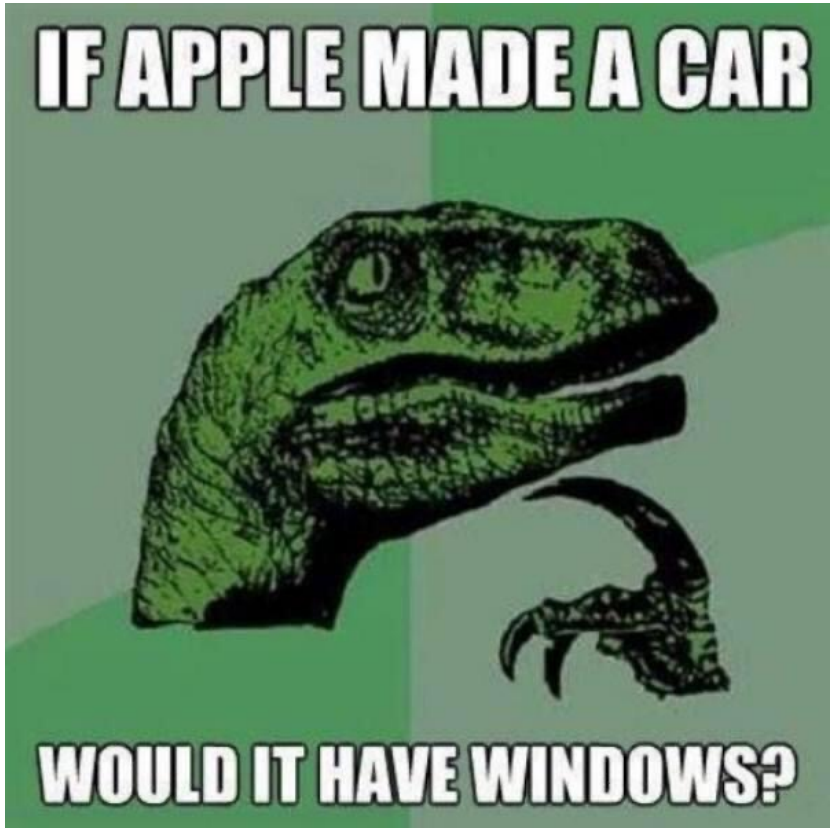
Loops

```
for i in range(10):
    print(i**3)
```

Very quick review of topics we have covered…

Conditional statements

```
num = 5
if num == 1:
    print("Monday")
elif num == 2:
    print("Tuesday")
elif num == 3:
    print("Wednesday")
elif num == 4:
    print("Thursday")
elif num == 5:
    print("Friday")
elif num == 6:
    print("Saturday")
elif num == 7:
    print("Sunday")
else:
    print("Invalid input")
```

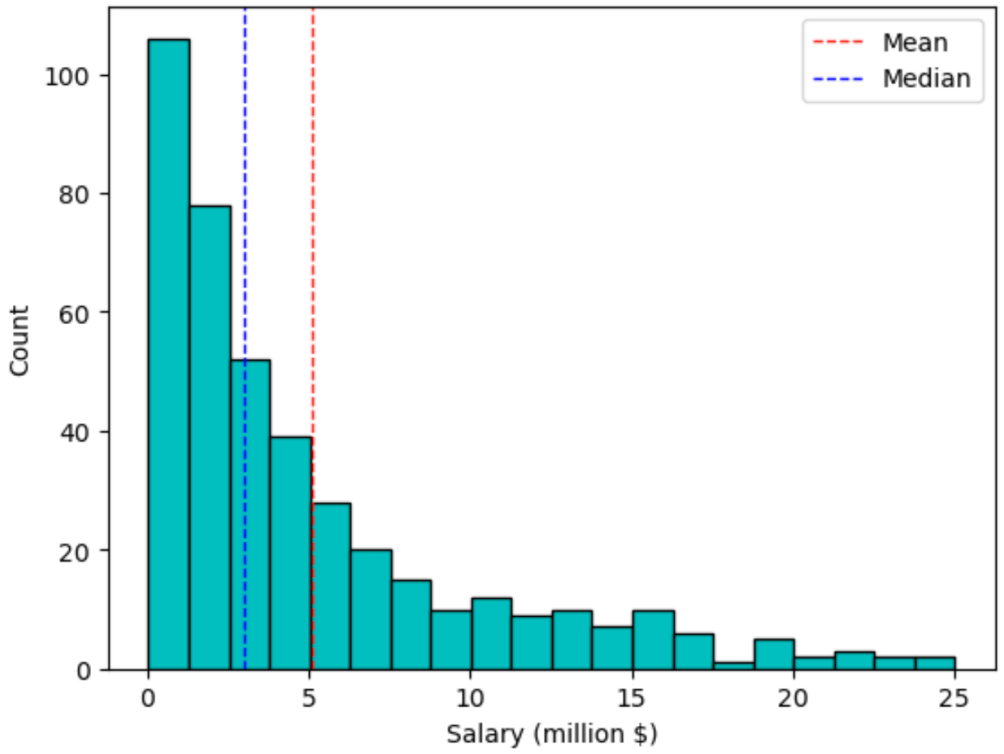


Very quick review of Python we have covered...

We have discussed statistics:

```
import statistics
statistics.median(data_list)
statistics.mean(data_list)
```

```
import matplotlib.pyplot as plt
plt.hist(data_list)
```



Questions?

Array computations

Arrays

Often we are processing data that is all of the same type

- For example, we might want to do processing on a data set of numbers
  - e.g., if we were just analyzing salary data

When we have data that is all of the same type, there are more efficient ways to process data than using a list

- i.e., methods that are faster and take up less memory

In Python, the *NumPy package* offers ways to store and process data that is all of the same type using a data structure called a ***ndarray***

There are also functions that operate on ndarrays that can do computations very efficiently.

ndarrays

We can import the NumPy package using: import numpy as np

We can then create an array by passing a list to the *np.array()* function

- my_array = np.array([1, 2, 3])
  - Note the textbook uses a function called make_array() that we will not use!

We can get elements of an array using similar syntax as using a list

- my_array[1] # what does this return

ndarrays have properties that tell us the type and size

- my_array.dtype # get the type of elements stored in the array
- my_array.shape # get the dimension of the array
- my_array.astype('str') # convert the numbers to strings

Let's explore this in Jupyter!

NumPy functions on numerical arrays

The NumPy package has a number of functions that operate very efficiently on numerical ndarrays

- np.sum()
- np.max(), np.min()
- np.mean(), np.median()
- np.diff() # takes the difference between elements
- np.cumsum() # cumulative sum

There are also "broadcast" functions that operate on all elements in an array

- my_array = np.array([12, 4, 6, 3, 4, 3, 7, 4])
- my_array * 2
- my_array + 7

Let's explore this in Jupyter!

Boolean arrays

It is often to compare all values in an ndarray to a particular value

- my_array = np.array([12, 4, 6, 3, 4, 3, 7, 4])
- my_array < 5 # any guesses what this will return
  - array([False, True, False, True, True, True, False, True])

This can be useful for calculating proportions

- True == 1 and False == 0
- Taking the sum of a Boolean array gives the total number of True values
- The number of True 's divided by the length is the proportion
  - Or we can use the np.mean() function

Categorical Variable

PLAYER	POSITION	TEAM	SALARY
str	str	str	f64
"Paul Millsap"	"PF"	"Atlanta Hawks"	18.671659
"Al Horford"	"C"	"Atlanta Hawks"	12.0
"Tiago Splitter..."	"C"	"Atlanta Hawks"	9.75625
"Jeff Teague"	"PG"	"Atlanta Hawks"	8.0
"Kyle Korver"	"SG"	"Atlanta Hawks"	5.746479

$$\text{Proportion centers} = \frac{\text{number of centers}}{\text{total number}}$$

Let's explore this in Jupyter!

Boolean masking

We can also use Boolean arrays to return values in another array

- This is called "Boolean masking" or "Boolean indexing"

```
my_array = np.array([12, 4, 6, 3])
boolean_mask = np.array([False, True, False, True, True])

smaller_array = my_array[boolean_mask]
```

This can be useful for calculating statistics on data that meet particular criteria:

- `np.mean(my_array[my_array < 5])  # what does this do?`

Boolean masking

Suppose you wanted to get the average salary of NBA players who were centers and you had these two ndarrays:

- Position: The position of all NBA players
- Salary: Their salaires

Could you do it?

Let's explore this in Jupyter!

Higher dimensional arrays

We can make higher dimensional arrays

- (matrices and tensors)

```
my_matrix = np.array([1, 2, 3], [4, 5, 6], [7, 8, 9])
my_matrix
```

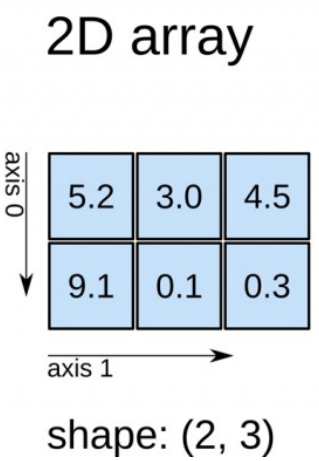
We can slice higher dimensional array

- my_matrix[0:2, 0:2]

We can apply operations to rows, columns, etc.

- np.sum(my_matrix, axis = 0) # sum the values down rows

Let's explore this in Jupyter!



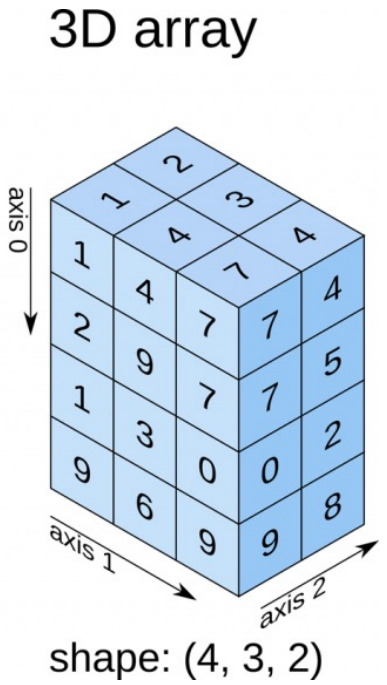
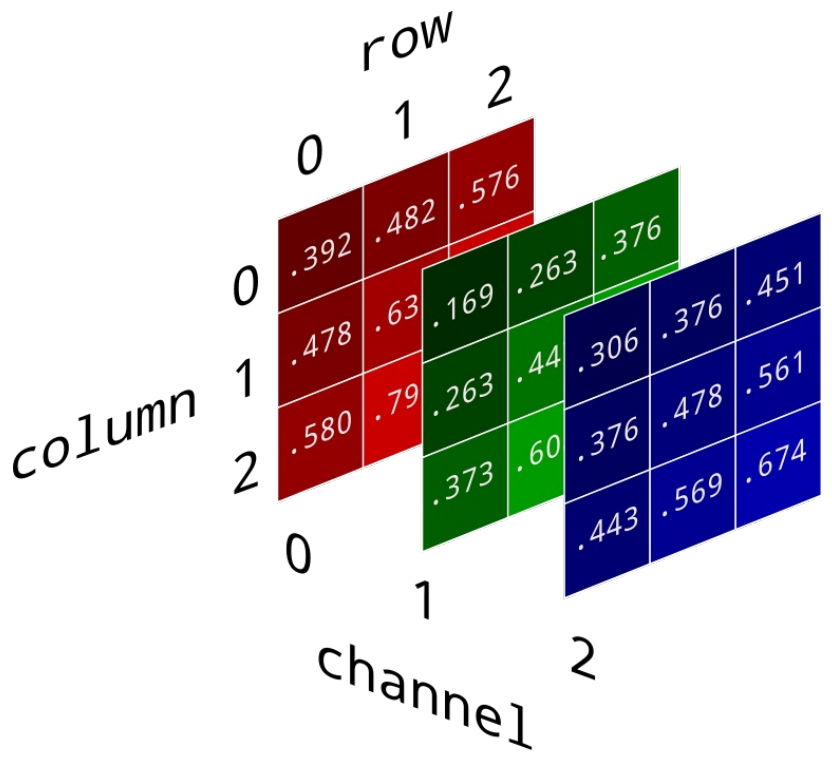


Image processing

3-dimemsional numerical arrays are often used to store digital images

We can use masking and other array operations to process images



Let's explore this in Jupyter!