YData: Introduction to Data Science

Class 06: Array computation continued

Overview

Quick review of:

- NumPy arrays
- Numerical computations
- Boolean masking

More numpy:

- Higher dimensional numerical arrays
- Image manipulation

Tuples and dictionaries

If there is time:

- Introduction to pandas Series and DataFrames

Announcement: Homework 2

Homework 2 is due on Gradescope on Sunday February 4th at 11pm

- **Be sure to mark each question on Gradescope!**

Notes:

- On problem 3, if the images are not showing up make sure to run the cells where the images are embedded
  - If you figure it out, help other people on Ed!

Quick review: ndarrays

The *NumPy package* efficiently tores and processes data that is all of the same type using ***ndarray***

```
import numpy as np

my_array = np.array([1, 2, 3])   # creating an ndarray
my_array[0]                      # accessing the 0th element

my_array.dtype              # get the type of elements
my_array.shape              # get the dimension
my_array.astype('str')      # convert to strings

sequential_nums = np.arange(1, 10)    # creates numbers 1 to 9
```

Quick review: functions on numerical arrays

The NumPy functions:

- 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_array2 = np.array([10, 9, 2, 8, 9, 3, 8, 5])
- my_array - my_array2

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
  - 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

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

Warm up: Number journey!

Please download the class 6 Jupyter notebook

- import YData
- YData.download.download_class_code(6)

Please complete the following number journey in the class 6 notebook:

- **Step 1**: Create an ndarray called *my_array* that has the numbers: 12, 4, 6, 3, 4, 3, 7, 4
- **Step 2:** Create an array *my_array2* that consists of the values of *my_array* minus the mean value of *my_array*.
- **Step 3:** Create *my_array3* which is a Boolean array that has True values for the positive values in *my_array2*
- **Step 4:** Calculate and print the total number of True values in *my_array3*

Let's take a number journey now…

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, 1])
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

If you had these two ndarrays:

- **Position**: The position of all NBA players
- **Salary**: Their salaries

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
```

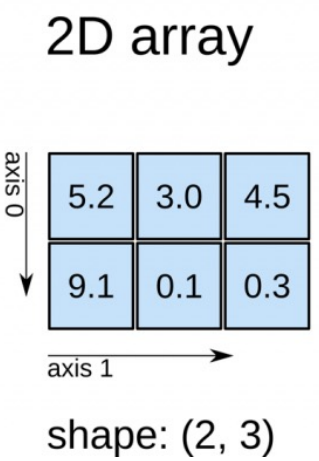
We can slice higher dimensional array

- 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!



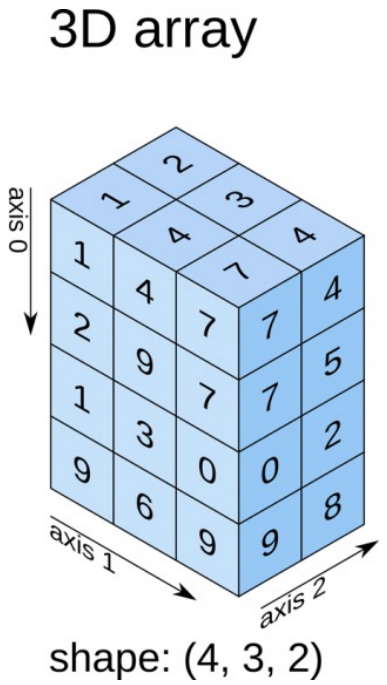
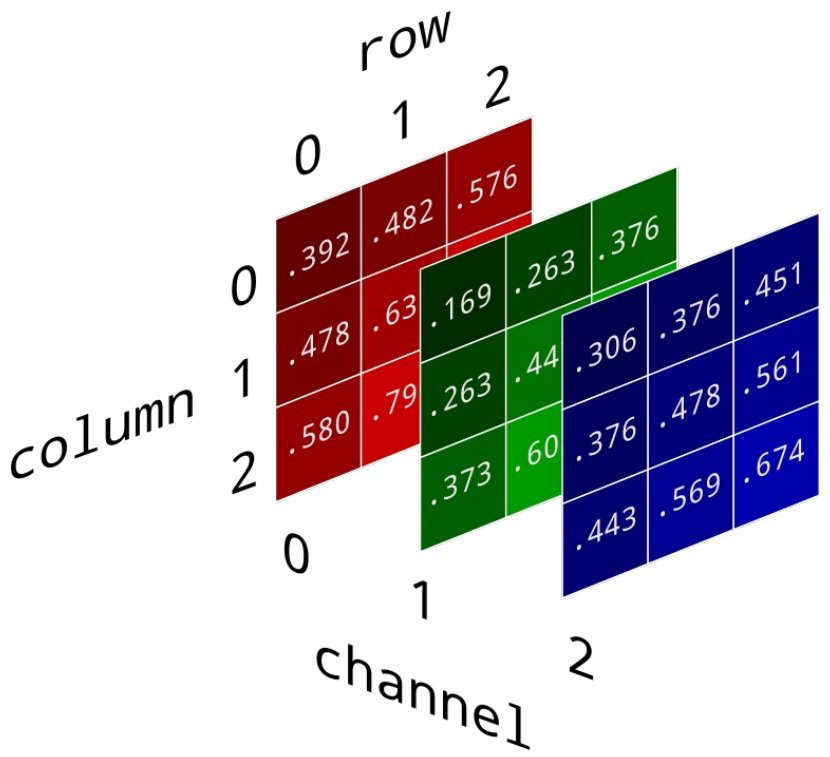


Image processing

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

- RGB image = Red, Green, Blue matrices

We can use masking and other array operations to process images



Let's explore this in Jupyter!

Tuples and Dictionaries

Tuples

Tuples are like lists but they are immutable; i.e., once they are created we can’t change the values in a tuple.

We can create a tuple using:

- my_tuple = (10, 20, 30)

Like lists, we can access elements of tuples using square brackets

- my_tuple[1]

We can’t change values in tuples:

- my_tuple[1] = 50 # Error!!!

Tuples

We can assign values in tuples into regular names using “tuple unpacking”

- my_tuple = (10, 20, 30)
- val1, val2, val3 = my_tuple
- val3

Let's explore this in Jupyter!

Dictionaries

Dictionaries allow you to look up ***values*** based on a ***key***

- i.e., you supply a “key” and the dictionary returns the stored value

We can create dictionaries using the syntax:

- my_dict = { 'key1': 5, 'key2': 20}

We can retrieve dictionary values by supplying a key using square brackets []

- my_dict['key2']

Let's explore this in Jupyter!

Series and Tables

Pandas: Series and DataFrames

"pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language."

There are two main data structures in pandas:

- **Series**: represent one-dimensional data
- **DataFrames**: represent data tables
  - i.e., relational data

pandas Series

pandas Series are: One-dimensional ndarray with axis labels

- (including time series)

Example: egg _prices

DATE	
1980-01-01	0.879
1980-02-01	0.774
1980-03-01	0.812

Index

values

pandas Series

We can access row elements by Index ***name*** using **.loc**

- egg_prices.loc["1980-01-01"]

We can access row elements by Index ***number*** using **.iloc**

- egg_prices.iloc[0]

Let's explore this in Jupyter!

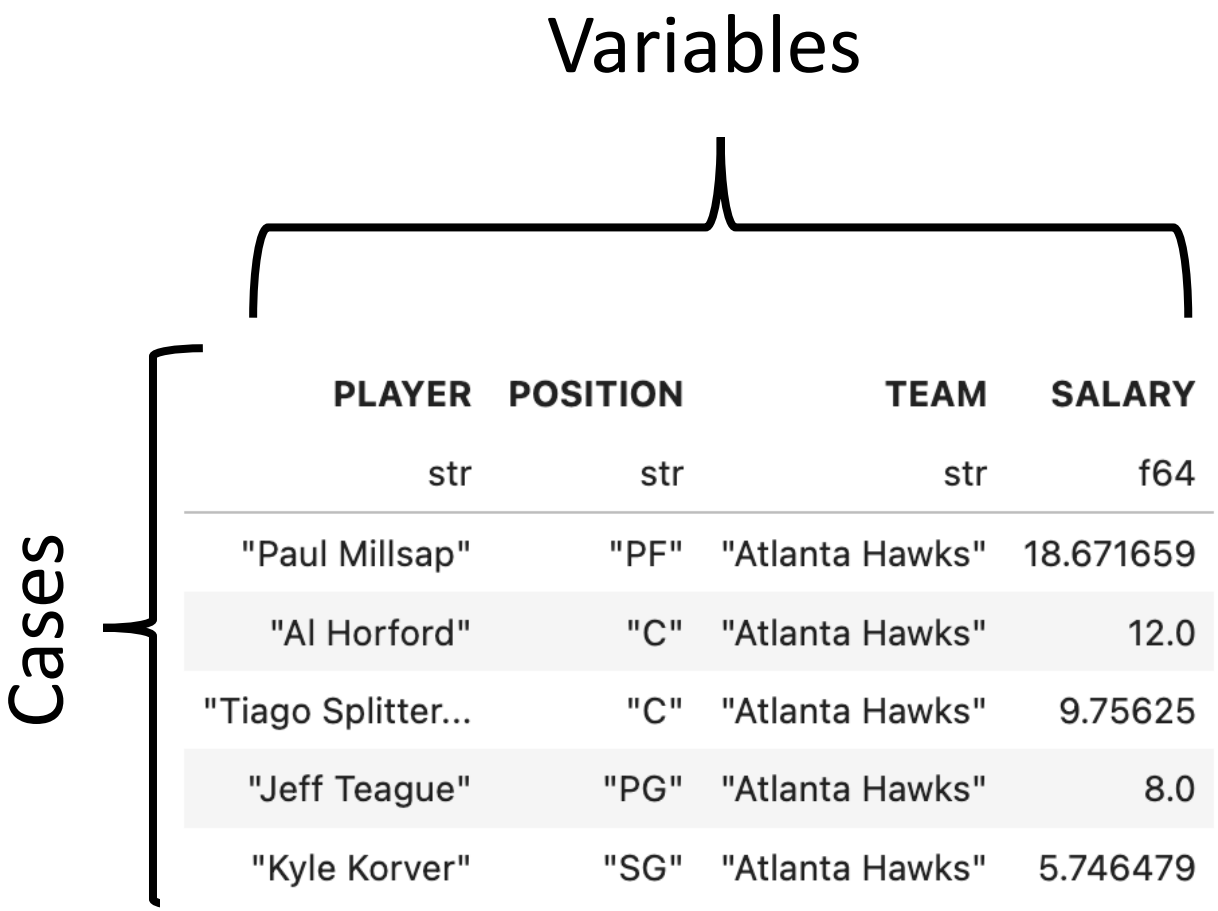
pandas DataFrames

Pandas DataFrame hold Table data

This is one of the most useful formats to extract insights from datasets

Often we read data into a DataFrame using:

- pd.read_csv("file.csv")



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

Let's explore this in Jupyter!

Selecting columns from a DataFrame

We can select a column from a DataFrame using square brackets:

- my_df[“my_col”] # returns a Series!

We can select multiple columns from a DataFrame by passing a list into the square brackets

- my_df[[“col1”, “col2”]]

Let's explore this in Jupyter!

Extracting rows from a DataFrame

We can extract rows from a DataFrame by:

1. The position they appear in the DataFrame
2. The Index values

We use the .iloc[] property to extract values by ***position***

- my_df.iloc[0]

We use the .loc[] property to extract values by ***Index value***

- my_df.loc[“index_name”]

Let's explore this in Jupyter!

Sorting rows from a DataFrame

We can sort values in a DataFrame using .sort_values("col_name")

- my_df.sort_values(“col_name”)

We can sort from highest to lowest by setting the argument ascending = False

- my_df.sort_values(“col_name”, ascending = False)

Let's explore this in Jupyter!

Adding new columns to a DataFrame

We can add a column to a data frame using square backets. For example:

- my_df[“new_col”] = values_array
- my_df["new col"] = my_df["col1"] + my_df["col2"]`

Let's explore this in Jupyter!

Creating aggregate statistics by group

There are several ways to get statistics by group

- Most methods use the .groupby() and .agg() methods

Perhaps the most useful way is to use the syntax:

```
my_df.groupby("group_col_name").agg(
      new_col1 = ('col_name', 'statistic_name1’),
      new_col2 = ('col_name', 'statistic_name2’),
      new_col3 = ('col_name', 'statistic_name3')
)
```

Let's explore this in Jupyter!

Next class: pandas continued...