YData: Introduction to Data Science

Class 05: Array computation continued

Overview

Brief review of:

- Statistics and visualizations of quantitative data

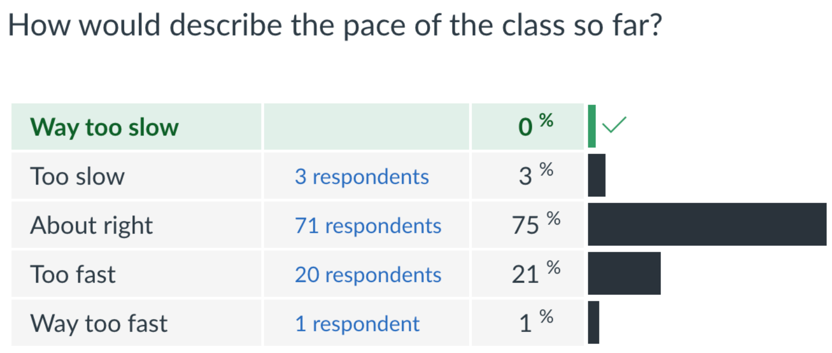
Continuation of numpy:

- Review: NumPy arrays
- Numerical array computations
- Higher dimensional numerical arrays
- Image manipulation

If there is time:

- Introduction to pandas Series and DataFrames

Announcement: Homework 2



How would describe the pace of the class so far?

Way too slow		0 %	✓
Too slow	3 respondents	3 %	
About right	71 respondents	75 %	
Too fast	20 respondents	21 %	
Way too fast	1 respondent	1 %	

Homework 2 has been posted!

It 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!
- I think the problems are at a reasonable level of difficulty
  - I will get your feedback…

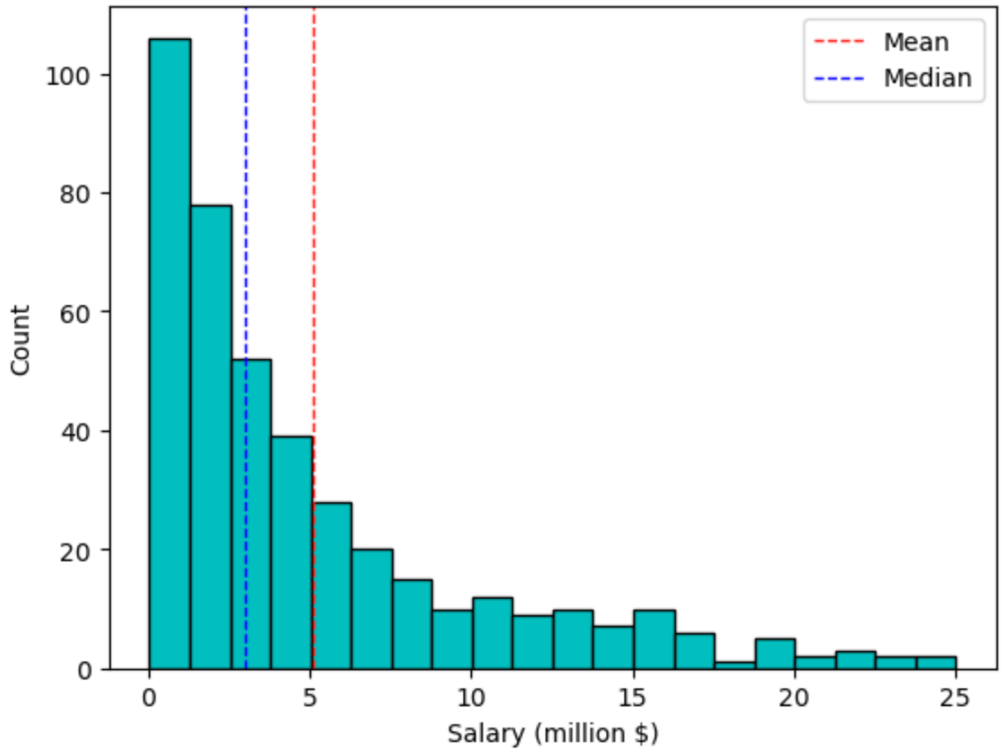
Questions?

Review of array computations

Very quick review: quantitative data

We can visualize quantitative data using histograms

Two statistics that measure “central value” are the ***mean*** and the ***median***



```
import matplotlib.pyplot as plt
plt.hist(data)
plt.xlabel("salary (million $)")
plt.ylabel("count")

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

Very quick review: outliers

Q: What is an **outlier?**

- A: An observed value that is notably distinct from the other values in a dataset

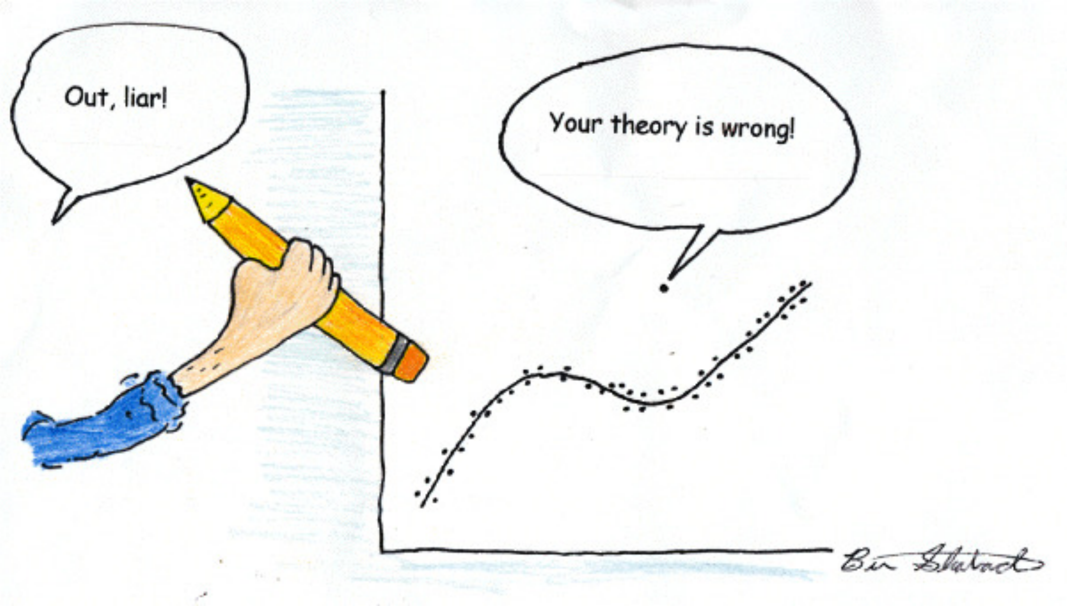
Q: Why are they problematic?

- A: can potentially have a large influence on the statistics you calculate

Q: What should you do if you have an outlier in your data?

A: See if you can understand what is causing it!

- If it's an error, delete the point
- If it's a real value, make sure it is not having a big effect on your conclusions, and/or use resistant statistics



Q: Is the mean and/or median resistant?

- A: The median is resistant while the mean is not

Review of NumPy arrays and functions

Processing data that is all of the same type can be more efficient than processing data of mixed types

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

The package also contains functions that operate efficiently on these arrays

Review: ndarrays

Let’s review this in Jupyter!

```
import numpy as np

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

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

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

NumPy 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

Let's explore this in Jupyter!

Boolean arrays

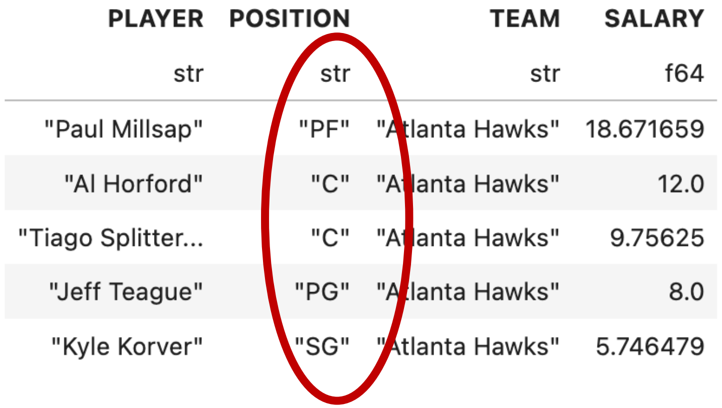
It is often to compare all values in an ndarray to a particular value

- 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, 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

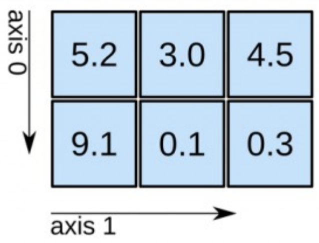
- `my_matrix[0:2, 0:2]`

We can apply operations to rows, columns, etc.

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

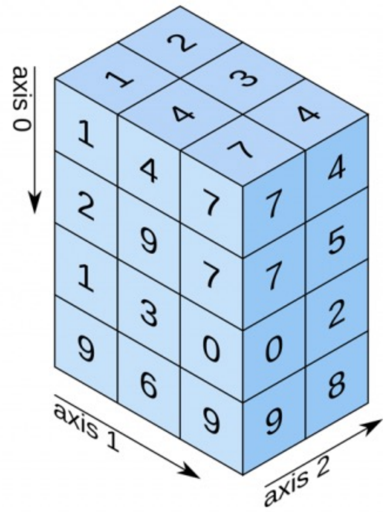
Let's explore this in Jupyter!

2D array



shape: (2, 3)

3D array



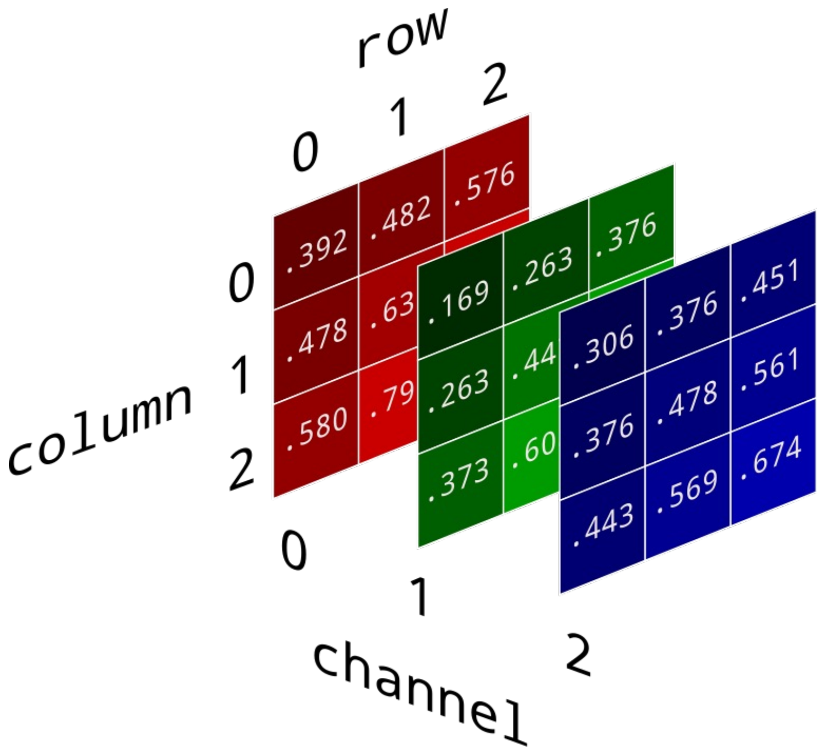
shape: (4, 3, 2)

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!

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 elements by Index ***name*** using **.loc**

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

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

- egg_prices.iloc[0]

Let's explore this in Jupyter!

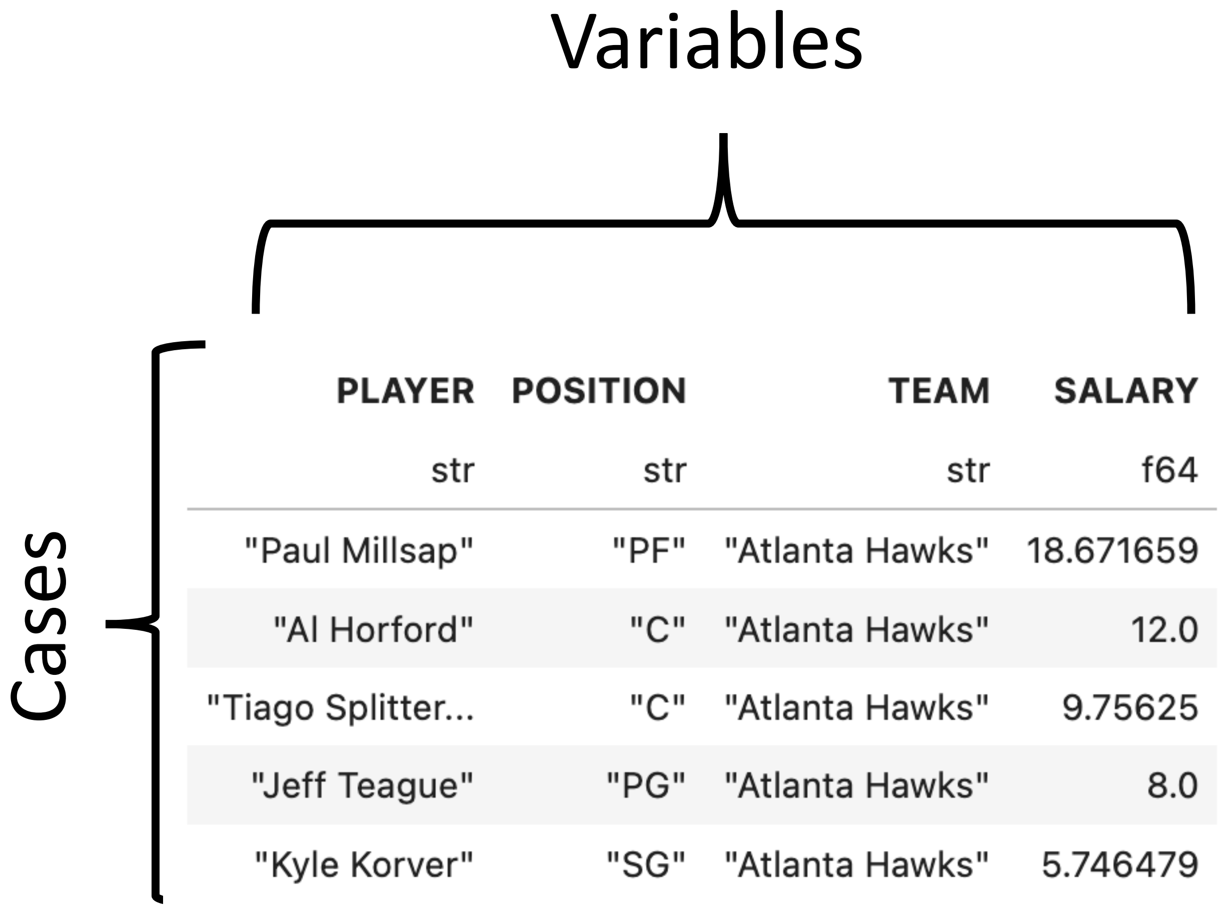
pandas DataFrames

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!

Next class: pandas continued…