

# **Pandas and Numpy**

**Welcome back to CS 2100!**

**Prof. Rasika Bhalerao**

# **Data Science**

**Effectively storing, manipulating, and analyzing collections of data**

# POV you're a data scientist who needs an example for how to do "logistic regression" in Python

The screenshot shows the scikit-learn API documentation for the `LogisticRegression` class. The left sidebar lists various modules and classes, with `sklearn.linear_model` expanded to show `LogisticRegression`. The main content area displays the API documentation for `LogisticRegression`, featuring a code example in a dark-themed code block:

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.linear_model import LogisticRegression
>>> X, y = load_iris(return_X_y=True)
>>> clf = LogisticRegression(random_state=0).fit(X, y)
>>> clf.predict(X[:2, :])
array([0, 0])
>>> clf.predict_proba(X[:2, :])
array([[9.82e-01, 1.82e-02, 1.44e-08],
       [9.72e-01, 2.82e-02, 3.02e-08]])
>>> clf.score(X, y)
0.97
```

Below the code example, there is a note: "For a comparison of the LogisticRegression with other classifiers see: [Plot classification probability](#)". At the bottom right, there is a link "[source]".

# **Python's two built-in categories of sequential types:**

## **Containers (list, tuple)**

**Store references to objects stored externally**

- Takes more space to store -- each element is a separate object with its own header containing metadata
- Iteration is slower -- Python jumps between memory locations to access each element

## **Flat Sequences (str, bytes, array)**

**Raw value elements stored internally within the sequence**

- Stored in consecutive memory locations
- Elements MUST be same type
- Fast iteration
- Efficient multi-dimensional data (matrices)

**Good for math and data processing**

# Arrays: efficient for storing numbers

```
from array import array  
  
a = array("l", [1, 2, 3])  
  
print(type(a))  
print(a[0])
```

```
<class 'array.array'>  
1
```

Arrays are built in to Python

# NumPy Arrays:

Container format for passing data between compatible scientific computing libraries such as SciPy and Pandas

```
import numpy as np

v1 = np.array([
    [1, 2, 3],
    [4, 5, 6]
])

print(v1)
print("v1 Python type:", type(v1))
print("NumPy array type:", v1.dtype)
print("v1 as python list:", v1.tolist())
```

You may need to `pip install numpy`

# Numpy (Numerical Python)

- a library that provides data structures and algorithms for numeric applications
- developed in 2005 by Travis Oliphant by combining Numeric and Numarray projects

# Live-coding: get the sum of **n** number of dice rolls

Hint: generate 20 (4x5) random integers between a lower (inclusive) and upper (exclusive) bound like this:

```
from typing import Any
import numpy as np

nums: np.ndarray[Any, np.dtype[np.int64]] =
    np.random.randint(10, 20, size=(4,5))

print(nums)
```

```
[[10 13 17 10 19]
 [13 15 11 12 11]
 [13 10 16 17 19]
 [13 18 12 14 12]]
```

# Poll: What is this function doing?

```
def something(n: int) -> np.ndarray:  
    matrix = np.random.randint(n, size=(n, n))  
    return np.mean(matrix, axis=1)
```

1. It creates a vector of length 1 (just one number) which is the average of  $n$  random numbers between 0 and  $n$
2. It creates a vector of length  $n$ , and each element in it is the average of  $n$  random numbers between 0 and  $n$
3. It creates a vector of length 1 which is the average of  $n$  squared random numbers between 0 and  $n$
4. It creates a vector of length  $n$ , and each element in it is a random number between 0 and  $n$

# Reshaping vectors

```
import numpy as np  
  
a = np.arange(6)  
print(a)  
  
print("\nReshape")  
a.shape = (2, 3)  
print(a)  
  
print("\nTranspose")  
print(a.transpose())
```

```
[0 1 2 3 4 5]
```

```
Reshape  
[[0 1 2]  
 [3 4 5]]
```

```
Transpose  
[[0 3]  
 [1 4]  
 [2 5]]
```

Doing this using Python lists would be a nightmare!

**POV you want to add two lists of numbers, but elementwise rather than concatenating them**

```
v1 = np.array([  
    [1, 2, 3],  
    [4, 5, 6]  
)  
  
v2 = np.array([  
    [7, 8, 9],  
    [0, 0, 0]  
)  
  
print(v1 + v2)
```

```
[[ 8 10 12]  
 [ 4  5  6]]
```

**List comprehension is nice... but not as efficient as NumPy arrays**



turning NumPy  
arrays into  
Python objects  
for iteration  
or loops



taking advantage  
of CPython's  
super fast built-in  
functions

# Adding arrays where only one of the side lengths match

```
v1 = np.array([  
    [1, 2, 3],  
    [4, 5, 6]  
)  
  
v2 = np.array(  
    [7, 8, 9],  
)  
  
print(v1 + v2)
```

```
[[ 8 10 12]  
 [11 13 15]]
```

If two vectors being added have different dimensions, it will raise a `ValueError`.

The exception: if one of the vectors is 1D, with a length equal to the length of each row in the other vector, it will add that vector to each row (broadcasting).

## **Other operations which are different in NumPy arrays versus lists:**

<b>Operation</b>	<b>List</b>	<b>NumPy Array</b>
Adding	Concatenating	Elementwise addition
Multiplying	Repeatedly concatenating	
Length	Number of elements	

## **Other operations which are different in NumPy arrays versus lists:**

<b>Operation</b>	<b>List</b>	<b>NumPy Array</b>
Adding	Concatenating	Elementwise addition
Multiplying	Repeatedly concatenating	Elementwise multiplication
Length	Number of elements	

## **Other operations which are different in NumPy arrays versus lists:**

Operation	List	NumPy Array
Adding	Concatenating	Elementwise addition
Multiplying	Repeatedly concatenating	Elementwise multiplication
Length	Number of elements	Norm (distance from origin)

Another thing data scientists do often:

## Work with spreadsheets of data

But using `open('file.csv', 'r')` to iterate through the file and read it line-by-line is a common but tedious thing -- there must be a library

# Pandas: a library for manipulating tabular data (spreadsheets)

- developed by Wes McKinney in 2008
- name derived from the term Panel Data
- designed to perform data analysis and visualisation
- uses NumPy arrays to store underlying data

You may need to:

1. pip install pandas
2. pip install pandas-stubs

Main Pandas data types:

- Series: one-dimensional labelled arrays of any data type
- DataFrames: two-dimensional labelled arrays of any data type
- Time series: index at data times
- Panel: three-dimensional container of data

# Creating a basic dataframe in Pandas

```
import pandas as pd

df = pd.DataFrame(
    {'Person': ['Elephant', 'Cat'],
     'Age': [13, 10]})
print(df)
```

	Person	Age
0	Cat	10
1	Elephant	13

# Read in a dataframe that is stored in a CSV file

**CSV = a spreadsheet stored as Comma-Separated Values**

data.csv:

```
Person,Age  
Cat,10  
Elephant,13
```

Code to read it as a dataframe:

```
df = pd.read_csv('/path/to/data.csv')
```

Code to put a dataframe into a new CSV:

```
df.to_csv('new_file.csv')
```

## Exercise:

```
data = {  
    'title': ['The Matrix', 'Inception', 'Interstellar', 'Blade Runner'],  
    'year': [1999, 2010, 2014, 1982],  
    'rating': [8.7, 8.8, 8.6, 8.1]  
}  
df = pd.DataFrame(data)
```

Let's:

- get the average rating
- print a sentence like "The Matrix was released in 1999" for each movie
- add a new movie "Dune" with year=2021 and rating=8.0

## Common Pandas operations: selecting a column

```
print(list(df['Age'])) # [10, 13]  
print(df['Age'])
```

```
0    10  
1    13  
Name: Age, dtype: int64
```

# Common Pandas operations: iterating through rows

```
for index, row in df.iterrows():
    print(f'Row number {index}:\n{row}\n')
```

Row number 0:  
Person Cat  
Age 10  
Name: 0, dtype: object

Row number 1:  
Person Elephant  
Age 13  
Name: 1, dtype: object

# Common Pandas operations: iterating through rows

Each row is a dict

```
for index, row in df.iterrows():
    print(f"{row['Person']}'s age is {row['Age']}")
```

```
Cat's age is 10
Elephant's age is 13
```

# Common Pandas operations: adding more rows

Create another dataframe and use `concat()` to concatenate them

```
new_rows = pd.DataFrame({  
    'Person': ['Dog', 'Giraffe'],  
    'Age': [3, 6]})  
  
df = pd.concat([df, new_rows], ignore_index=True)  
print(df)
```

	Person	Age
0	Cat	10
1	Elephant	13
2	Dog	3
3	Giraffe	6

# Common Pandas operations: adding more columns

```
df['BFF'] = ['Cat', 'Giraffe', 'Cat', 'Elephant']
print(df)
```

	Person	Age	BFF
0	Cat	10	Cat
1	Elephant	13	Giraffe
2	Dog	3	Cat
3	Giraffe	6	Elephant

(BFF = best friend forever)

We will do more Pandas operations (like sorting and filtering) in Lecture 15.

## Poll: How can I find the age of the oldest person in `df` ?

1. `df(max['Age'])`
2. `max(df['Age'])`
3. `max(row['Age'] for _, row in df.iterrows())`
4. `max('Age' for _, row in df.iterrows())`

# Raising Errors

Like this `ZeroDivisionError`:

```
def int_divide(num1: int, num2: int) -> int:  
    """Returns the result of dividing num1 and num2, rounded to the nearest int"""  
    if num2 == 0:  
        raise ZeroDivisionError  
    return round(num1 / num2)  
  
print(int_divide(7, 3)) # 2
```

# Raising Errors

If there is not a built-in error which fits our needs, we can create our own:

```
class CatPettingError(Exception):
    """Custom exception raised when cat petting has gone wrong"""
    def __init__(self, message: str = "The cat did not like that."):
        super().__init__(message)

def pet_cat() -> None:
    if random() < 0.3:
        raise CatPettingError
    print('purrr')

for i in range(5):
    pet_cat()
```

# Raising Errors

Only two differences between our custom exception and the built-in errors:

- Its name (`CatPettingError`)
- The message it prints when raised ("The cat did not like that.")
  - useful if we want to give the client a more useful hint as to what went wrong

## Exercise

Let's write a function that writes a given dataframe to a given file, but only if that file does not already exist. If it exists, then it should raise a `FileExistsError`.

## Solution:

```
import os

def safe_to_csv(df: pd.core.frame.DataFrame, filename: str) -> None:
    if os.path.exists(filename):
        raise FileExistsError
    df.to_csv(filename)
```

Another tool used in HW:

## The OS module

Particularly useful:

- `os.path.exists("path/to/file")` checks if the file exists
- `os.path.join('base/path', 'filename')` safely combines the base path and the filename to create a `str` holding the path to the file
- `os.listdir('path/to/directory')` lists the files in the directory

Because the function `open('filename', 'w')` overwrites the file if it already exists, these functions in the `os` module can help us to prevent overwriting precious files.

## **Poll:**

- 1. What is your main takeaway from today?**
- 2. What would you like to revisit next time?**