Popular Libraries

CPSC 217: Introduction to Computer Science for Multidisciplinary Studies I
Winter 2023

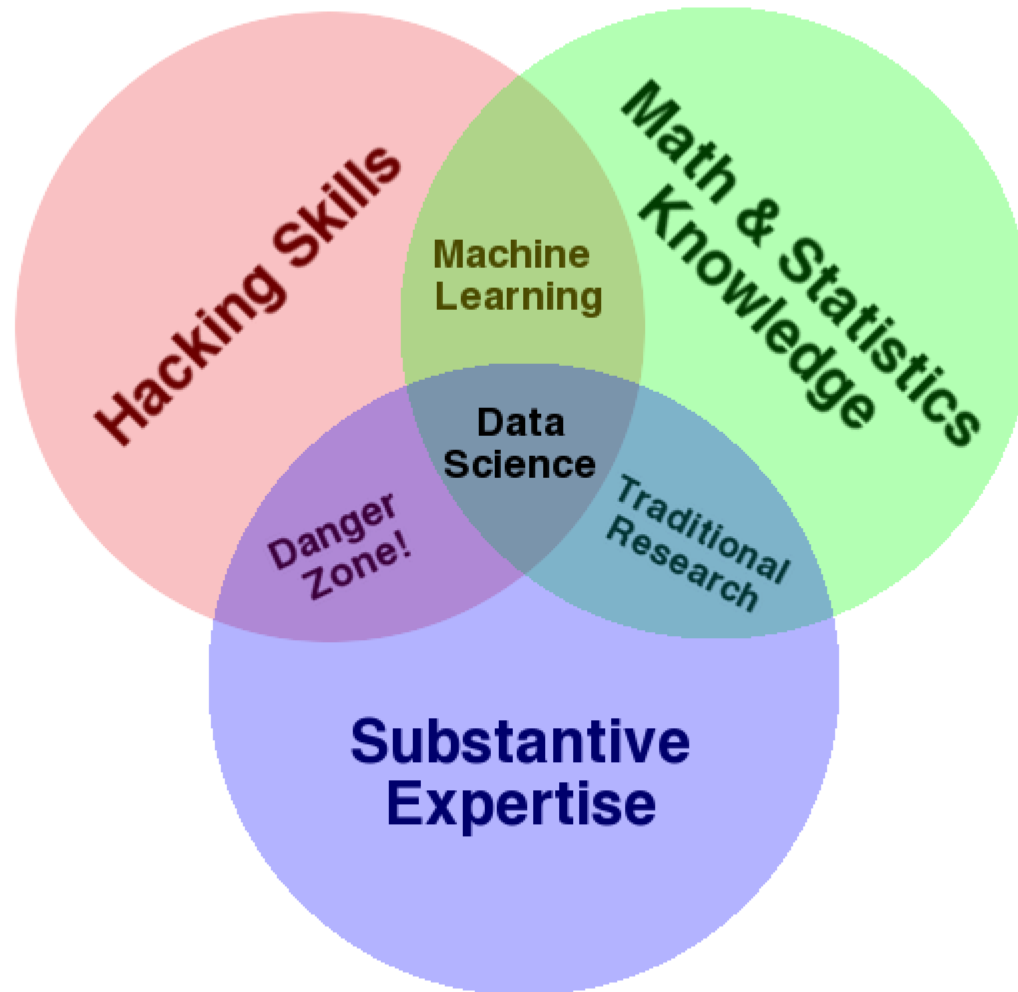
Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

January 9, 2023

Copyright © 2023

UNIVERSITY OF CALGARY

What is Data Science?



Drew Conway's September 2010 Data Science Venn Diagram
http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram

UNIVERSITY OF CALGARY

Python

- **Python** 3 – Most popular Data Science programming language
  - **R** is popular from the Statistics side of things, **Julia** in Mathematics

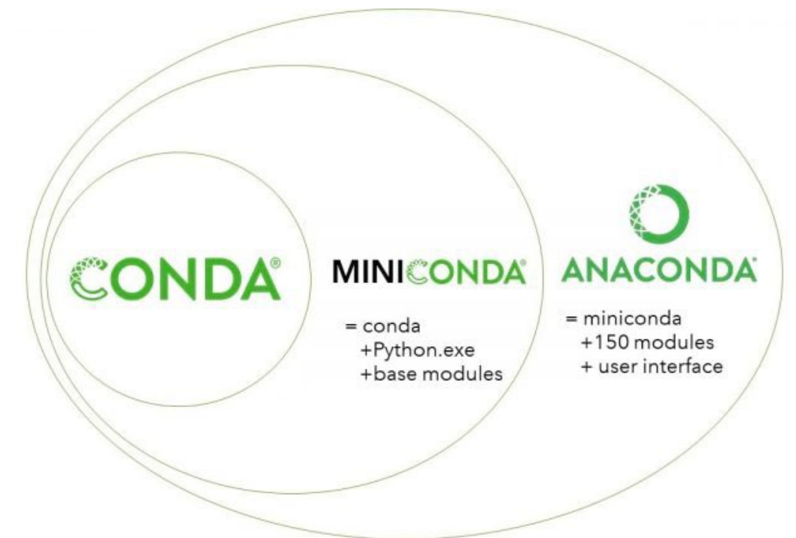
1. Very strong support of packages, tutorials, and knowledge
2. Ease of integrating more efficient languages behind scenes like **C++**, etc.
3. Often prototype in **R**/others but implement in **Python** for final production

1. Interpreted, so can be slow (unless break out to **C++** like with ***numpy***)
2. Not built for multi-thread concurrency without effort (unless break out like with ***tensorflow*** for neural networks to use multi-core/GPU)

Data Science Installation Method (managed)

- **Anaconda** – Distribution of **Python** 3 and package manager that allows easy access to most popular data science libraries
- **Miniconda** – Lighter weight than **Anaconda** as it doesn't pre-download as many packages
  - Generally early learning path is to get miniconda
  - https://docs.conda.io/en/latest/miniconda.html
  - Then install

```
1 conda install numpy pandas matplotlib seaborn ipython jupyterlab
2 conda install scipy scikit-learn tensorflow keras statsmodels
```



The first is for Data Science tasks while the second line of installation is for Machine Learning

UNIVERSITY OF CALGARY

Data Science Installation (lightweight)

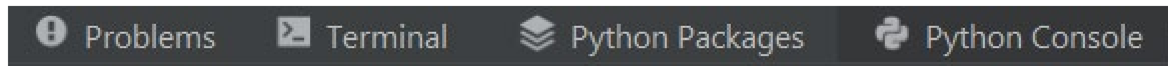
- Could just use available **Jupyter** notebook ***ipython***-like environments such as

1. **Google Colaboratory** https://colab.research.google.com/ (public)
2. **Syzygy** https://ucalgary.syzygy.ca/ (university member host)

- Or setup your local install of **Python**
  - IDE choice up to you (for Python prefer I like **Pycharm**)
  - Recommend virtual environment (good for UofC lab machines)
  - **pip** for packages (python package manager)

Data Science Installation (IDE-Pycharm)

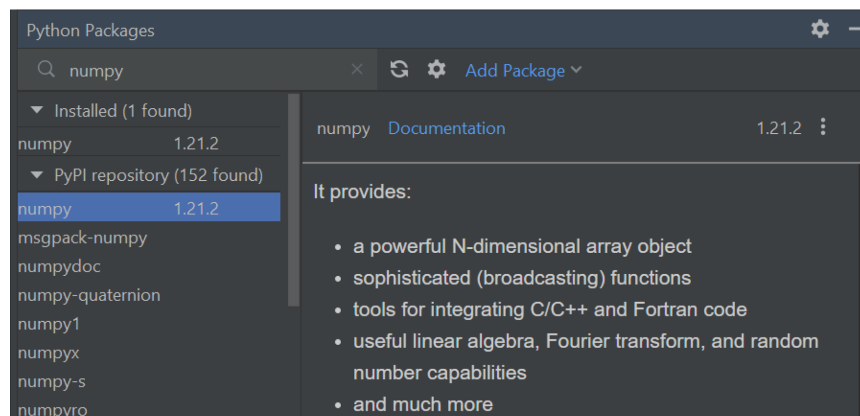
- Bottom bar of **Pycharm**



- Can use terminal to install packages to main IDE **Python** install

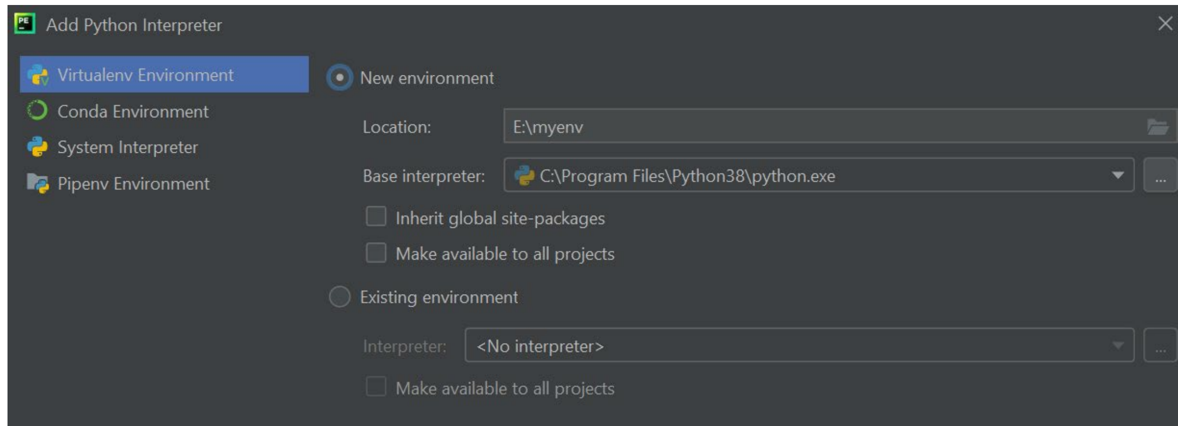
```
Terminal:   Local
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\API> pip install numpy
```

- Can also use package manager to do same

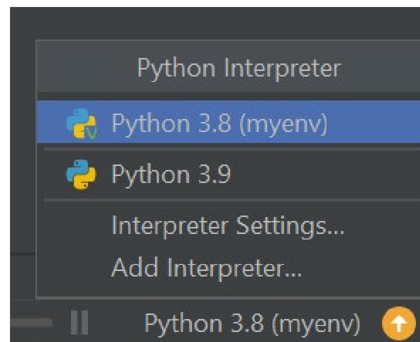


Data Science Installation (IDE-Pycharm)

- Instead of using system interpreter you can create virtual environment



- Bottom right corner you can toggled between **Python** interpreters



Data Science Installation (windows shell)

```
1 python -m venv myenv
2
3 .\myenv\Scripts\activate
4
5 pip install numpy
6 pip install pandas
7 pip install matplotlib
8 pip install seaborn
9 pip install ipython jupyterlab notebook
10
11 pip install scipy scikit-learn tensorflow keras statsmodels
12
13 deactivate
```


UNIVERSITY OF CALGARY

Data Science Installation (unix shell)

```
1 python -m venv myenv
2
3 source myenv/bin/activate
4
5 pip install numpy
6 pip install pandas
7 pip install matplotlib
8 pip install seaborn
9 pip install ipython jupyterlab notebook
10
11 pip install scipy scikit-learn tensorflow keras statsmodels
12
13 deactivate
```

Packages - Interaction

IPython (2001)

- https://ipython.org/
- This is a interactive version of **Python**. Differences from regular interactive is that it can store state for reference using In and Out blocks (stored as arrays), and you can also break out of it to run shell commands from inside (like to install library or make folders and manage files)

jupyter notebooks (2014)

- https://jupyter.org/
- GUI to **IPython**,. Your computer hosts a execution kernel and you access it using a web browser web-based interface (like the **Google Colaboratory** and **Syzygy** options given earlier) Can do other languages as well.

Packages – Standard Data Science

- ***numpy*** – manipulation of homogenous array-based data, container
  - https://numpy.org/

- ***pandas*** – 2010, heterogenous and labeled data like tables or databases, brings spreadsheets like functionality to data handling (on top of ***numpy***)
  - https://pandas.pydata.org/

- ***matplotlib*** – publication quality visuals
  - https://matplotlib.org/

- ***seaborn*** – better charts on top of ***matplotlib*** (compete with **R**)
  - https://seaborn.pydata.org/

Packages – Machine Learning

- ***scipy*** – ***scikit*** uses for algorithms, good at storing sparse, common scientific computing tasks (integration, linear algebra, optimization, signal processing, statistics distributions/variables/etc.)
  - https://scipy.org/
- ***scikit-learn*** – 2010 machine learning, classification, regression, clustering, etc.
  - https://scikit-learn.org/stable/index.html
- ***tensorflow*** – 2015 open-sourced distributed neural network library, 2019 v2.0
  - https://www.tensorflow.org/
- ***keras*** – 2015 deep learning library (often on top of ***tensorflow***)
  - https://keras.io/
- ***statmodels*** – classical statistics and econometrics
  - https://www.statsmodels.org/stable/index.html

Importing Libraries in Python (style)

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 import tensorflow as tf
6 import statsmodels as sm
```


UNIVERSITY OF CALGARY

numpy, pandas, matplotlib

- numpy - > Numpy is one of the main libraries used in machine learning and data science: it is used for a variety of mathematical computations, written in optimized C code at its base.

- pandas -> Pandas is a library for data manipulation. It is highly compatible with Numpy, although it has some subtleties.

- matplotlib -> As it name implies, this is Python’s main plotting library. A number of other libraries such as seaborn rely on this.

numpy

UNIVERSITY OF CALGARY

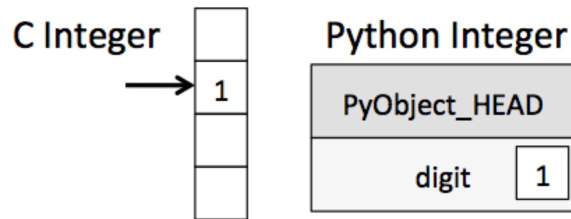
numpy

- **Num**erical **Py**thon library (2005 brought together disparate library ideas)
- More efficient data and storage operations as arrays grow larger
- **Python** integer is more than an integer, and a list more than just values

UNIVERSITY OF CALGARY

integers?

- **Python** integer is more than an integer
- **Python** 3.4 integer



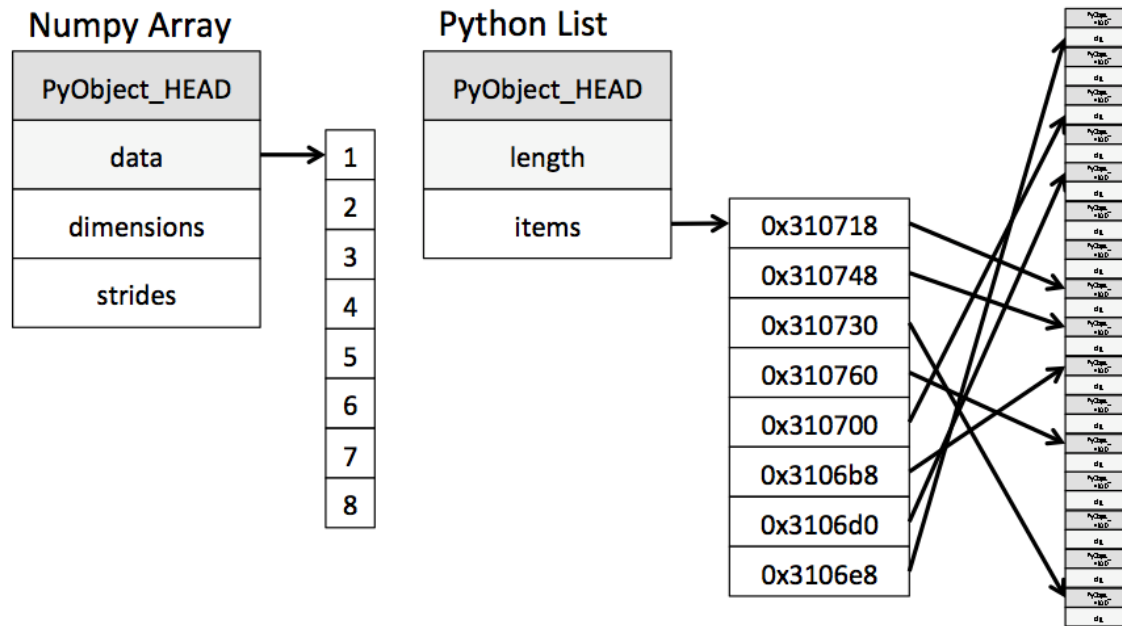

```
1 struct _longobject {
2   long ob_refcnt;
3   PyTypeObject *ob_type;
4   size_t ob_size;
5   long ob_digit[1];
6 }
```

- A **C** integer is essentially a label for a position in memory whose bytes encode an integer value.
- A **Python** integer is a pointer to a position in memory containing all the Python object information, including the bytes that contain the integer value.

UNIVERSITY OF CALGARY

lists?

- A **Python** list is more than just a list of values



- A **Python** list contains a pointer to a block of pointers, each of which in turn points to a full **Python** object like the **Python** integer we saw earlier.
- **numpy** arrays are a single pointer to a block of contiguous data.

UNIVERSITY OF CALGARY

numpy

- **Num**erical **Py**thon library
- More efficient data and storage operations as arrays grow larger
- **Python** integer is more than an integer, and a list more than just values
- **numpy** arrays allow us to put data all in one place and drastically improve the our ability to manipulate it quickly.
- In essences what **numpy** does is provide a portal from **Python** through to **C** implementations of storage arrays, allowing us to access the strengths of that language in ways not normally available in **Python**.

UNIVERSITY OF CALGARY

Create

- Numerous options to create **numpy** arrays

```
[ ] import numpy as np

[5] np.array(range(5))
array([0, 1, 2, 3, 4])

[12] np.array([1,2,3,4,5], dtype="float32")
array([1., 2., 3., 4., 5.], dtype=float32)

[11] np.zeros(10, dtype=int)
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

[13] np.full(3, 10)
array([10, 10, 10])

[16] np.random.random(3)
array([0.65720754, 0.97030252, 0.77293412])
```


UNIVERSITY OF CALGARY

Datatypes

- To gain the power of C arrays Python now has to care about all the types that a C programmer has to manage

Data type	Description
`bool_`	Boolean (True or False) stored as a byte
`int_`	Default integer type (same as C `long`; normally either `int64` or `int32`)
`intc`	Identical to C `int` (normally `int32` or `int64`)
`intp`	Integer used for indexing (same as C `ssize_t`; normally either `int32` or `int64`)
`int8`	Byte (-128 to 127)
`int16`	Integer (-32768 to 32767)
`int32`	Integer (-2147483648 to 2147483647)
`int64`	Integer (-9223372036854775808 to 9223372036854775807)
`uint8`	Unsigned integer (0 to 255)
`uint16`	Unsigned integer (0 to 65535)
`uint32`	Unsigned integer (0 to 4294967295)
`uint64`	Unsigned integer (0 to 18446744073709551615)
`float_`	Shorthand for `float64`.
`float16`	Half precision float: sign bit, 5 bits exponent, 10 bits mantissa
`float32`	Single precision float: sign bit, 8 bits exponent, 23 bits mantissa
`float64`	Double precision float: sign bit, 11 bits exponent, 52 bits mantissa
`complex_`	Shorthand for `complex128`.
`complex64`	Complex number, represented by two 32-bit floats
`complex128`	Complex number, represented by two 64-bit floats

UNIVERSITY OF CALGARY

Attributes

- Three attributes of all arrays 1.the dimensions (count) 2. tuple of size of each dimension AKA shape 3. Total size which is product of tuple.

```
np.random.seed(0)  # seed for reproducibility
x3 = np.random.randint(10, size=(3, 4, 5))  # Three-dimensional array

print(x3)
print("x3 ndim: ", x3.ndim)
print("x3 shape:", x3.shape)
print("x3 size: ", x3.size)
```

```
x3 ndim:  3
x3 shape: (3, 4, 5)
x3 size:  60
```

```
[[[5 0 3 3 7]
  [9 3 5 2 4]
  [7 6 8 8 1]
  [6 7 7 8 1]]

 [[5 9 8 9 4]
  [3 0 3 5 0]
  [2 3 8 1 3]
  [3 3 7 0 1]]

 [[9 9 0 4 7]
  [3 2 7 2 0]
  [0 4 5 5 6]
  [8 4 1 4 9]]]
```


UNIVERSITY OF CALGARY

Indexing

- Mostly slight notation adjustment for multiple dimensions [a][b] is now [a,b]
- Slicing also retained (but slices are NOT copies! [use .copy() to copy])

```
x1 = np.array([5, 0, 3, 3, 7, 9])
print(x1[4])
print(x1[-1])
print(x1[-2])
x2 = np.array([[3, 5, 2, 4],
       [7, 6, 8, 8],
       [1, 6, 7, 7]])
print(x2[2, 0])
print(x2[2, -1])
x2[0, 0] = 12
print(x2)
```

```
7
9
7
1
7
[[12  5  2  4]
 [ 7  6  8  8]
 [ 1  6  7  7]]
```


UNIVERSITY OF CALGARY

Reshape, Concatenate, Stack, Split

```
x = np.array([1, 2, 3])
y = x.reshape((3, 1))
print(x)
print(y)
z = np.array([3, 2, 1])
w = np.concatenate([x, z])
u = np.stack([x,z])
print(w)
print(u)
a, b = np.split(w, [3])
print(a)
print(b)
```

```
[1 2 3]
[[1]
 [2]
 [3]]
[1 2 3 3 2 1]
[[1 2 3]
 [3 2 1]]
[1 2 3]
[3 2 1]
```


UNIVERSITY OF CALGARY

Speed

- numpy operations speed comes from a large variety of Universal Functions (Ufuncs) which are ‘vectorized operations’ designed to run at higher speed than if we let Python’s loops manage things
- Both computer same answer!

```
[48] import numpy as np
     np.random.seed(0)

     def compute_reciprocals(values):
         output = np.empty(len(values))
         for i in range(len(values)):
             output[i] = 1.0 / values[i]
         return output

     values = np.random.randint(1, 10, size=5)

     print(compute_reciprocals(values))
     print(1.0 / values)
```

```
[0.16666667 1.         0.25       0.25       0.125     ]
[0.16666667 1.         0.25       0.25       0.125     ]
```

Speed

- Both computer same answer!
- Milliseconds versus microseconds (a difference in order of magnitude of 1000 here)

```
big_array = np.random.randint(1, 100, size=100000)
print("Python")
%timeit compute_reciprocals(big_array)
print("numpy")
%timeit (1.0 / big_array)
```

```
Python
1 loop, best of 5: 225 ms per loop
numpy
The slowest run took 4.95 times longer t
1000 loops, best of 5: 227 µs per loop
```

UFuncs

- Most **Ufuncs** are accessible using straight operators, although we can always use the longer form of np.divide for example
- Some don't have operators like

np.abs(array)

np.sin(array), np.cos(array), etc.

np.log2(array), np.ln(array), etc.

np.sqrt(array), np.floor(array), etc.

- Others available via **scipy** under **special**

```
1 from scipy import special
```

Operator	Equivalent ufunc	Description
`+`	`np.add`	Addition (e.g., `1 + 1 = 2`)
`-`	`np.subtract`	Subtraction (e.g., `3 - 2 = 1`)
`-`	`np.negative`	Unary negation (e.g., `-2`)
`*`	`np.multiply`	Multiplication (e.g., `2 * 3 = 6`)
`/`	`np.divide`	Division (e.g., `3 / 2 = 1.5`)
`//`	`np.floor_divide`	Floor division (e.g., `3 // 2 = 1`)
`**`	`np.power`	Exponentiation (e.g., `2 ** 3 = 8`)
`%`	`np.mod`	Modulus/remainder (e.g., `9 % 4 = 1`)

Boolean arrays

Operator	Equivalent ufunc	Operator	Equivalent ufunc
`==`	`np.equal`	`!=`	`np.not_equal`
`<`	`np.less`	`<=`	`np.less_equal`
`>`	`np.greater`	`>=`	`np.greater_equal`

Operator	Equivalent ufunc	Operator	Equivalent ufunc
`&`	`np.bitwise_and`	`|`	`np.bitwise_or`
`^`	`np.bitwise_xor`	`~`	`np.bitwise_not`

- Can use booleans to ask questions, combine booleans with bitwise operators
- Will product boolean arrays which can then be used as 'masks' for filtering

```
x = np.array([5, 2, 3, 4, 1])
print(x < 3)
print(np.any(x<4))
print(np.all(x<4))
print(np.sum(x<4))
print(np.sum((x < 4) & (x > 1)))
print(np.sum((x < 2) | (x > 3)))
print(x[(x < 2) | (x > 3)])
```

```
[False  True False False  True]
True
False
3
2
3
[5 4 1]
```


UNIVERSITY OF CALGARY

Save/Load

- **numpy** has both straight forward save/load, but also ability to save multiple in a form that multiple arrays can be reloaded and referenced by stored key

```
array = np.random.randint(0, 100, size = 15)
np.save('filename', array)
temp = np.load("filename.npy")
print(temp)
array2 = np.random.randint(0, 100, size = 15)
np.savez('filename2', a=array, b=array2)
temp1 = np.load("filename2.npz")
print(temp1['a'])
print(temp1['b'])
```

```
[59 83  3  6 33 96 72 54 36 29 81 85 95 51 65]
[59 83  3  6 33 96 72 54 36 29 81 85 95 51 65]
[27  3 62 15  0 47 56 96 79 40 88 14 92 96 89]
```


UNIVERSITY OF CALGARY

pandas

pandas

- **pandas** (2010) is built on **numpy**
- recognition of outside influences and inside software engineering influences
  - Outside -> want array storage to work like spreadsheets/databases
  - Inside -> named data for readability and lookup
- Built around **Series** and **Dataframe** ideas which are essentially 1 dimensional array of data and multidimensional array of data with row/column labels
- Allows us to reference data with names, while maintaining **numpy** speed

Series

- 1D data is a Series, which acts like a specialized dictionary (default indices are integers)

```
import pandas as pd
print(pd.Series([0.25, 0.5, 0.75, 1.0]))
print(pd.Series([0.25, 0.5, 0.75, 1.0], index = ['a','b','c','d']))
data = pd.Series({'a':0.25, 'b':0.5, 'c':0.75, 'd':1.0})
print(data['a'])
```

```
0    0.25
1    0.50
2    0.75
3    1.00
dtype: float64
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
0.25
```


UNIVERSITY OF CALGARY

Dataframe

- 2D data is a DataFrame, which acts like a specialized dictionary of Series

```
data_1 = pd.Series({'a':0.25, 'b':0.5, 'c':0.75, 'd':1.0})
data_2 = pd.Series({'a':1, 'b':2, 'c':3, 'd':4})
data = pd.DataFrame({'data_1':data_1, "data_2":data_2})
print(data)
print(data.index)
print(data.columns)
print(data['data_1'])
```

```
   data_1  data_2
a    0.25       1
b    0.50       2
c    0.75       3
d    1.00       4
Index(['a', 'b', 'c', 'd'], dtype='object')
Index(['data_1', 'data_2'], dtype='object')
a    0.25
b    0.50
c    0.75
d    1.00
Name: data_1, dtype: float64
```

Danger Will Robinson!

- **Slices are not copies** (better thought of a views of data)
- This is great for speed as data isn't copied, but it means modification of slice means modifying the original data

```
data = pd.Series({'a':0.25, 'b':0.5, 'c':0.75, 'd':1.0})
print(data)
data['a':'c']['a'] = 5
print(data)
```

```
a    0.25
b    0.50
c    0.75
d    1.00
dtype: float64
a    5.00
b    0.50
c    0.75
d    1.00
dtype: float64
```


UNIVERSITY OF CALGARY

Indexing?

- We can index using the dictionary keys (index) but sometimes that can be confusing as there is both the explicit index and an always maintained integer (starts at 0) index as well
- **loc**, **iloc** is how we ensure we are doing the numerical type
- reset_index() useful for pushing current index into column and returning to integer index

```
data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])
print(data)
print(data[1])
print(data.loc[1])
print(data.iloc[1])
```

```
1    a
3    b
5    c
dtype: object
a
a
b
```

```
data = pd.Series(['a', 'b', 'c'], index=[1, 3, 5])
data = data.reset_index()
print(data)
```

```
   index  0
0      1  a
1      3  b
2      5  c
```


UNIVERSITY OF CALGARY

DataFrame operations

- **DataFrame** allow for very quick generation of new columns of data
- Can be transposed quickly and **values()** gives the **numpy** data

```
area = pd.Series({'California': 423967, 'Texas': 695662, 'New York': 141297, 'Florida': 170312, 'Illinois': 149995})
pop = pd.Series({'California': 38332521, 'Texas': 26448193, 'New York': 19651127, 'Florida': 19552860, 'Illinois': 12882135})
data = pd.DataFrame({'area':area, 'pop':pop})
print(data)
data['density'] = data['pop']/data['area']
print(data)
print(data.T)
print(data.values)
```

```
              area       pop
California  423967  38332521
Texas       695662  26448193
New York    141297  19651127
Florida     170312  19552860
Illinois    149995  12882135
              area       pop     density
California  423967  38332521   90.413926
Texas       695662  26448193   38.018740
New York    141297  19651127  139.076746
Florida     170312  19552860  114.806121
Illinois    149995  12882135   85.883763
           California         Texas      New York       Florida      Illinois
area     4.239670e+05  6.956620e+05  1.412970e+05  1.703120e+05  1.499950e+05
pop      3.833252e+07  2.644819e+07  1.965113e+07  1.955286e+07  1.288214e+07
density  9.041393e+01  3.801874e+01  1.390767e+02  1.148061e+02  8.588376e+01
```

DataFrame operations

- **Wary of missing ()**

```
print(data.loc[(data['density'] > 100) & (data['density'] < 125)])
print(data.loc[(data['density'] > 100) & data['density'] < 125])
```

```
            area       pop     density
Florida   170312  19552860  114.806121
              area       pop     density
California  423967  38332521   90.413926
Texas       695662  26448193   38.018740
New York    141297  19651127  139.076746
Florida     170312  19552860  114.806121
Illinois    149995  12882135   85.883763
```


UNIVERSITY OF CALGARY

DataFrame UFuncs

- **numpy Ufuncs** can be applied across a whole **DataFrame**

```python
import pandas as pd
import numpy as np
rng = np.random.RandomState(42)
ser = pd.Series(rng.randint(0, 10, 4))
df = pd.DataFrame(rng.randint(0, 10, (3, 4)),
                  columns=['A', 'B', 'C', 'D'])
print(ser)
print(np.exp(ser))
print(df)
print(np.sin(df * np.pi / 4))
```

```
0    6
1    3
2    7
3    4
dtype: int64
0     403.428793
1      20.085537
2    1096.633158
3      54.598150
dtype: float64
   A  B  C  D
0  6  9  2  6
1  7  4  3  7
2  7  2  5  4
          A             B         C             D
0 -1.000000  7.071068e-01  1.000000 -1.000000e+00
1 -0.707107  1.224647e-16  0.707107 -7.071068e-01
2 -0.707107  1.000000e+00 -0.707107  1.224647e-16
```

Capabilities

- Missing Data (None/NaN)
- Concatenate
- Join (like databases)
- Aggregate
- GroupBy
- Filtering, Transform, Apply, Map
- Pivot Tables (like excel)
- Describe, Correlation, Statistics
- Unique, Count, Masking

UNIVERSITY OF CALGARY

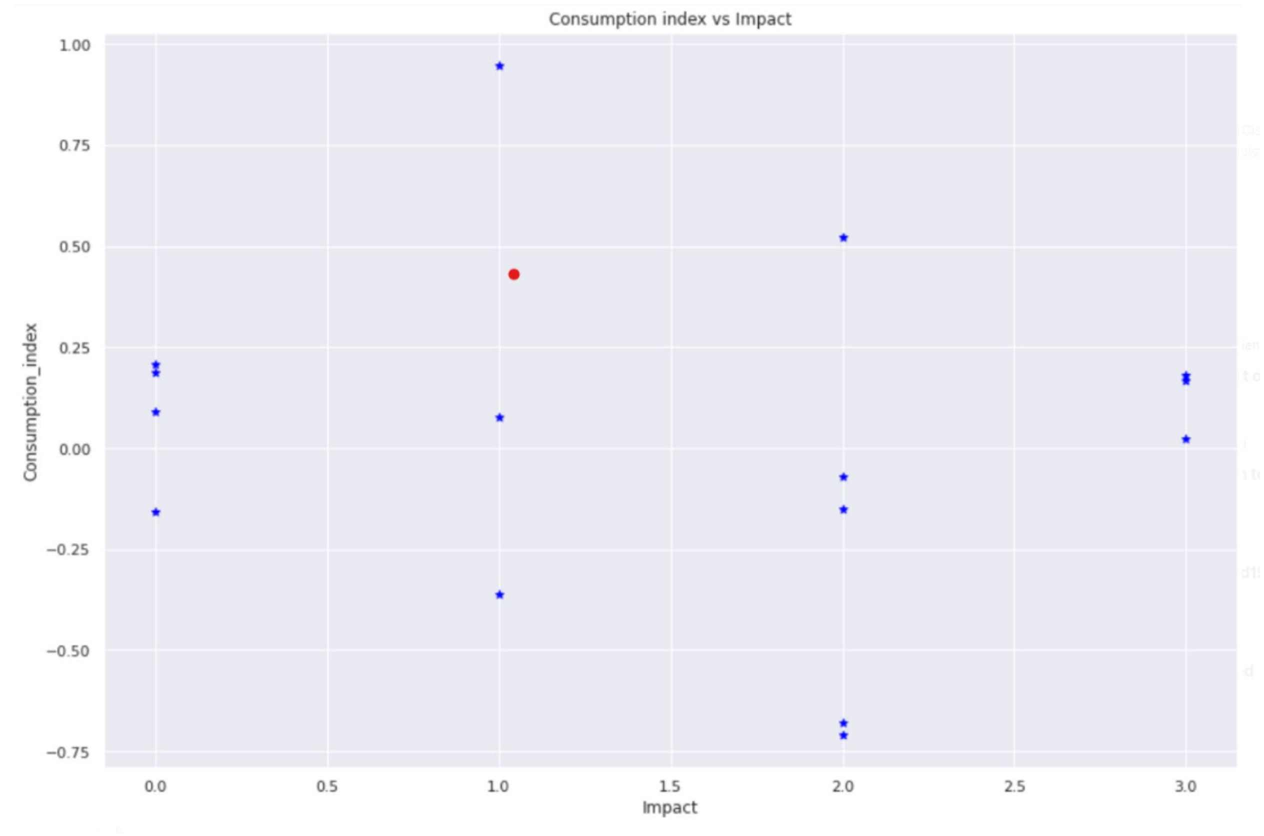
Reading/Writing Files

- Pandas can do many operations like
- read_csv, read_excel, read_html, read_json, read_pickle, read_sql
- In addition to others for data
- Possible to do many things such as setting names, skipping rows, limit rows, etc.
- to_csv used to write csv data
- Other libraries relevant here Python **csv** module, **json** library, **beautifulsoup/lxml/html5lib** for **HTML**, binary data using **pickle**, web-api generally use **request**, SQL database using **sqlite3/sqlalchemy**

matplotlib

matplotlib charts!

```
# specify figure to plot on
plt.figure(figsize=(15,10))
# Obtain x and y arrays
x = df2["Impact"]
y = df2["kll_index"]
# plot the scatterplot
plt.scatter(x,y, color="blue", marker="*")
# add title and labels
plt.title("Consumption index vs Impact")
plt.xlabel("Impact")
plt.ylabel("Consumption_index")
plt.show()
```

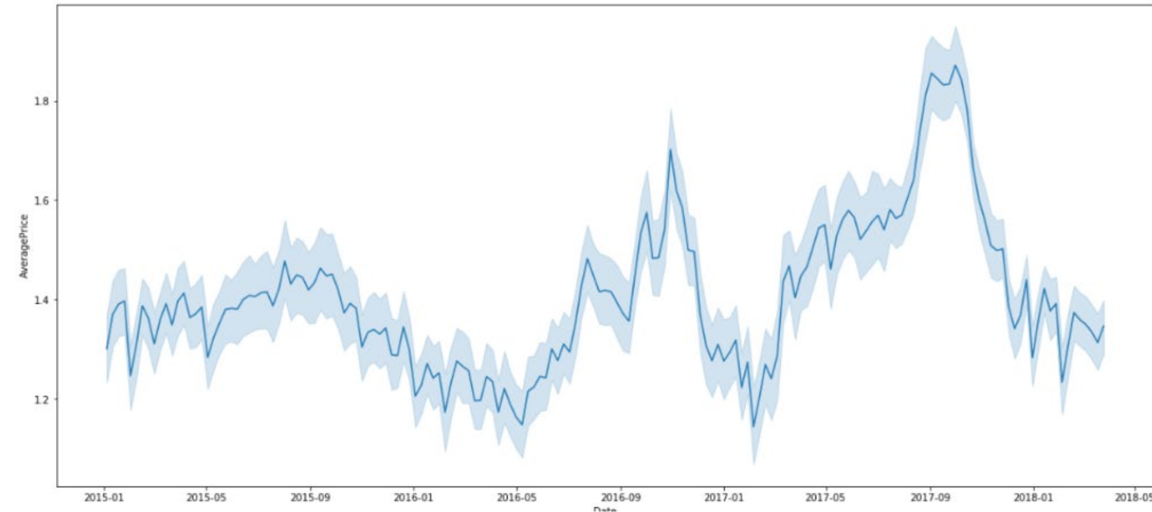


https://matplotlib.org/stable/users/getting_started/index.html

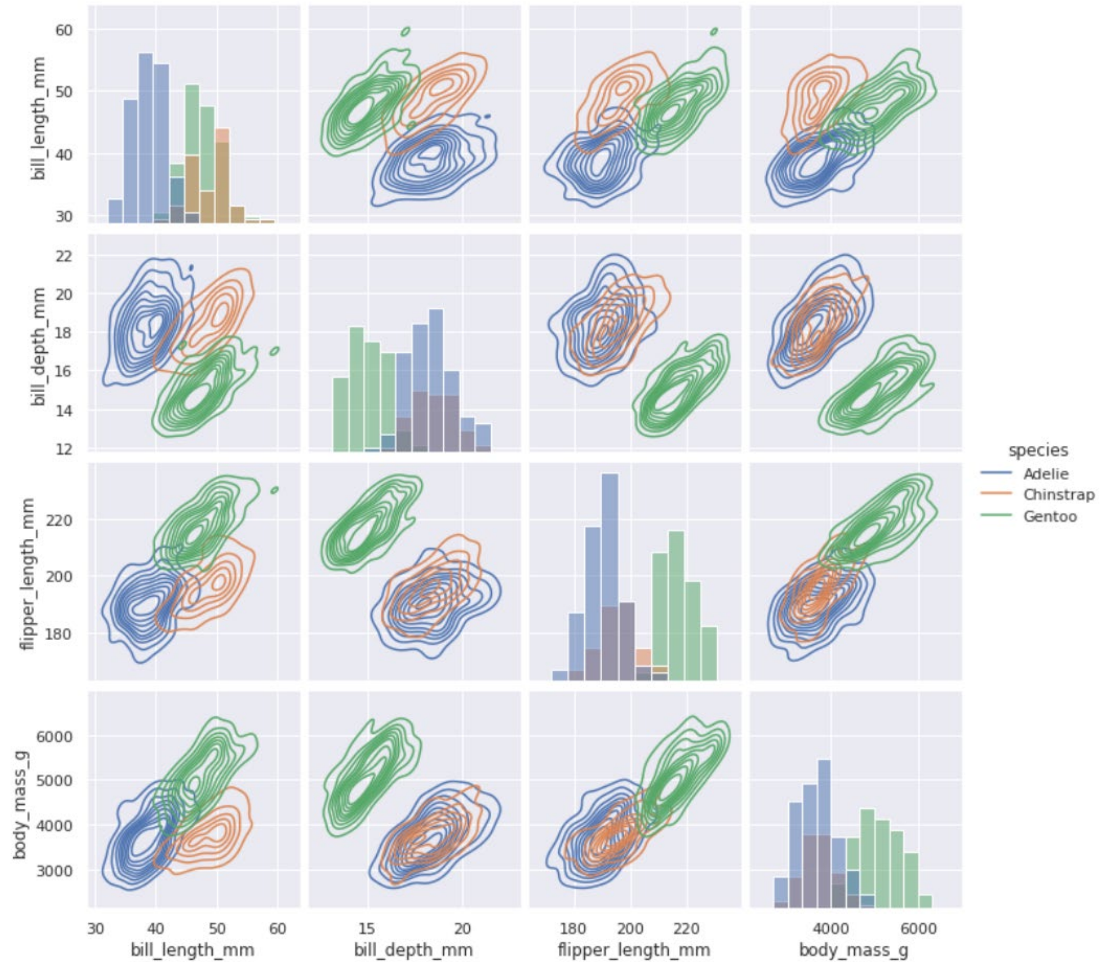
UNIVERSITY OF CALGARY

Seaborn?

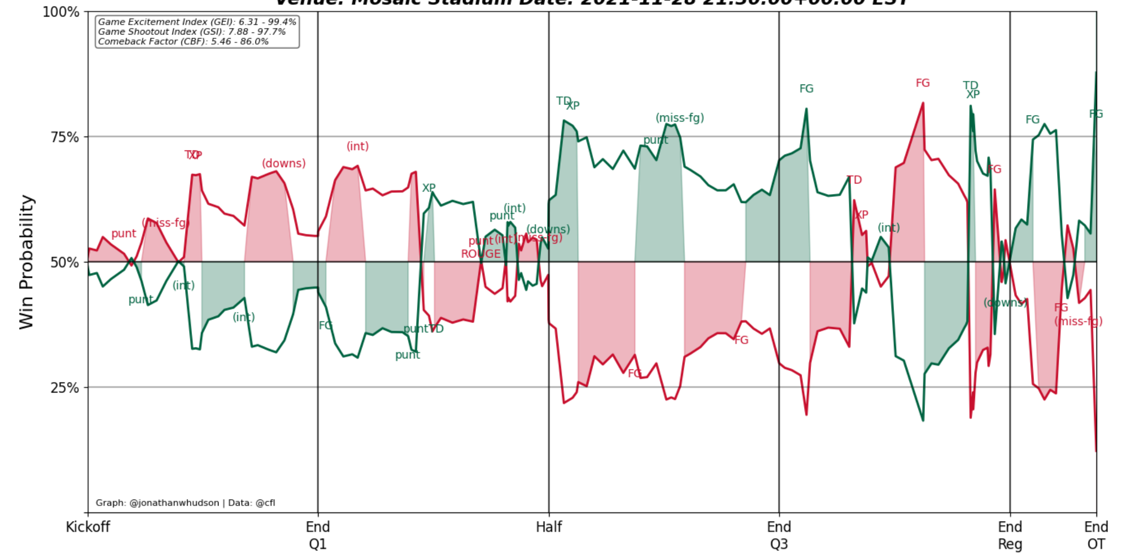
```
import seaborn as sns            #pip install seaborn

df = pd.read_csv('avocado.csv')'
plt.figure(figsize=(20,9)
sns.lineplot(data=df, x='Date',y='AveragePrice')
```



Seaborn?



Seaborn code Output (Image by author)



UNIVERSITY OF CALGARY

Onward to … History of Computer Science.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY