Module 1: Exploratory Data Analysis in One Dimension

Overview

- This module will introduce you to vectorized data structures and the Pandas library.
- You will learn how to explore single dimensional data sets.
- Pandas 2.x

Vectorized Data Structures let you perform operations on your data all at once

Advantages of Vectorized Data Structures

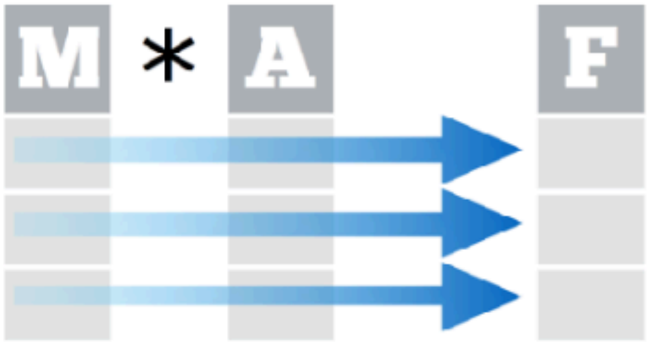
- More readable and simpler code
- Allows the module to optimize for CPU and Memory usage
- Pass serialized objects from one system to another. (More on this later)

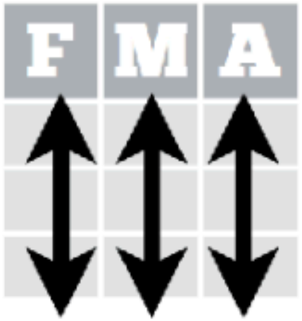
```
for x in range(0, len( data )):
    data[x] = data[x] + 1
```

```
for x in range(0, len( data )):
    data[x] = data[x] + 1
```

```
odds = evens + 1
```

F = M * A





Each **variable** is saved in its own **column**

&

Each **observation** is saved in its own **row**

No loops

Dimensions	Name	Description
1	Series	Indexed 1 dimensional data structure
1	Timeseries	Series using timestamps as an index
2	DataFrame	A two dimensional table

Columns

rows

Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

Libraries in Python

What is a Python library?

- Python libraries are also called modules
- Python has large set of built-in libraries
- Can contain classes, functions or variables
- Imported into a script or session

```
import ipaddress
```

Python library: full name

Functions defined in a **library** are called using **dot notation** when importing the full library.

library name + period + function name

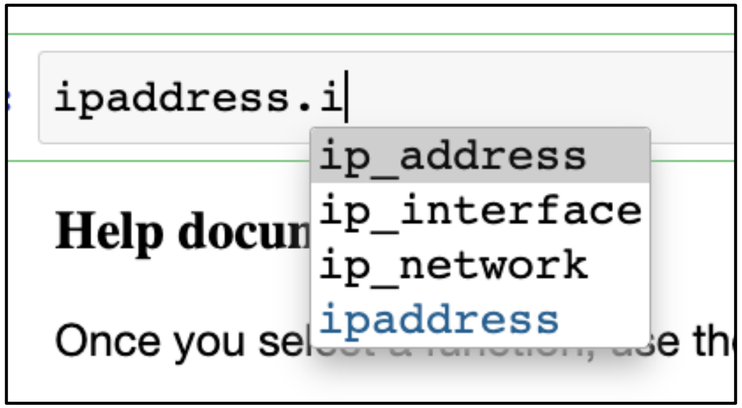
```
import ipaddress
user_address = ipaddress.ip_address("0:0:0:0:0:FFFF:185.15.58.226")
user_address.is_private
```

```
False
```

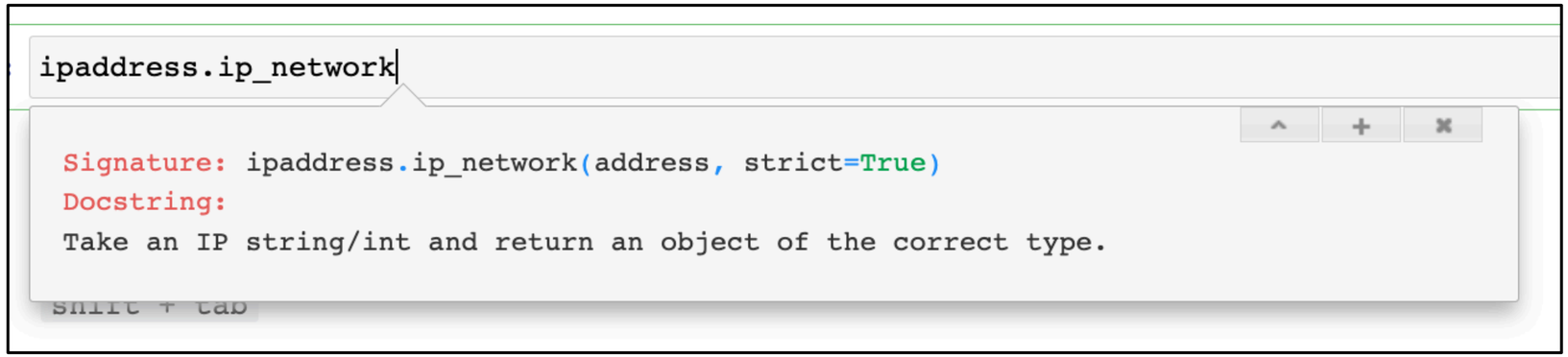
Note: try using use tab completion after typing the dot and the letter 'i' to get a tool tip of possible functions that start with 'i' from the ipaddress library.

Python library: Jupyter shortcuts

Tab completion



Shift + Tab = documentation



Python library: alias

Use an **alias** to avoid typing the full name of the **library** each time a **function** is called.

```
import ipaddress as ip
user_address = ip.ip_address("185.15.58.226")
```

```
import pandas as pd
raw_empty_data_frame = pd.DataFrame()
```

Python library: specific functions

When importing specific **functions** from a **library**, the function name can be called without the dot notation if we import them specifically using

```
from library import function1, function2
```

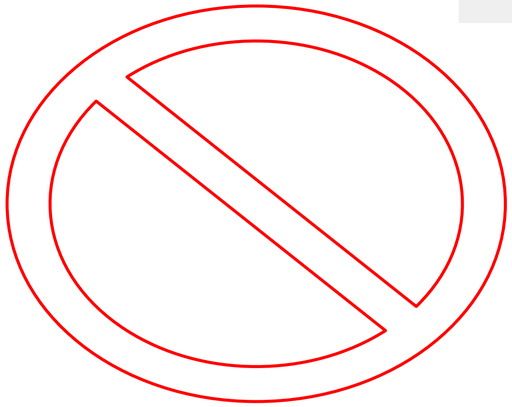
```
from ipaddress import ip_address, ip_network, ip_interface

ip_interface("0:0:0:0:0:FFFF:185.15.58.226")
```

Python library: wild card

DON’T DO THIS!!!

When importing specific **functions** from a **library**, all the functions, variables, etc can be imported using the **wildcard**.

```
from ipaddress import *
ip_network("192.0.2.0/24")
```

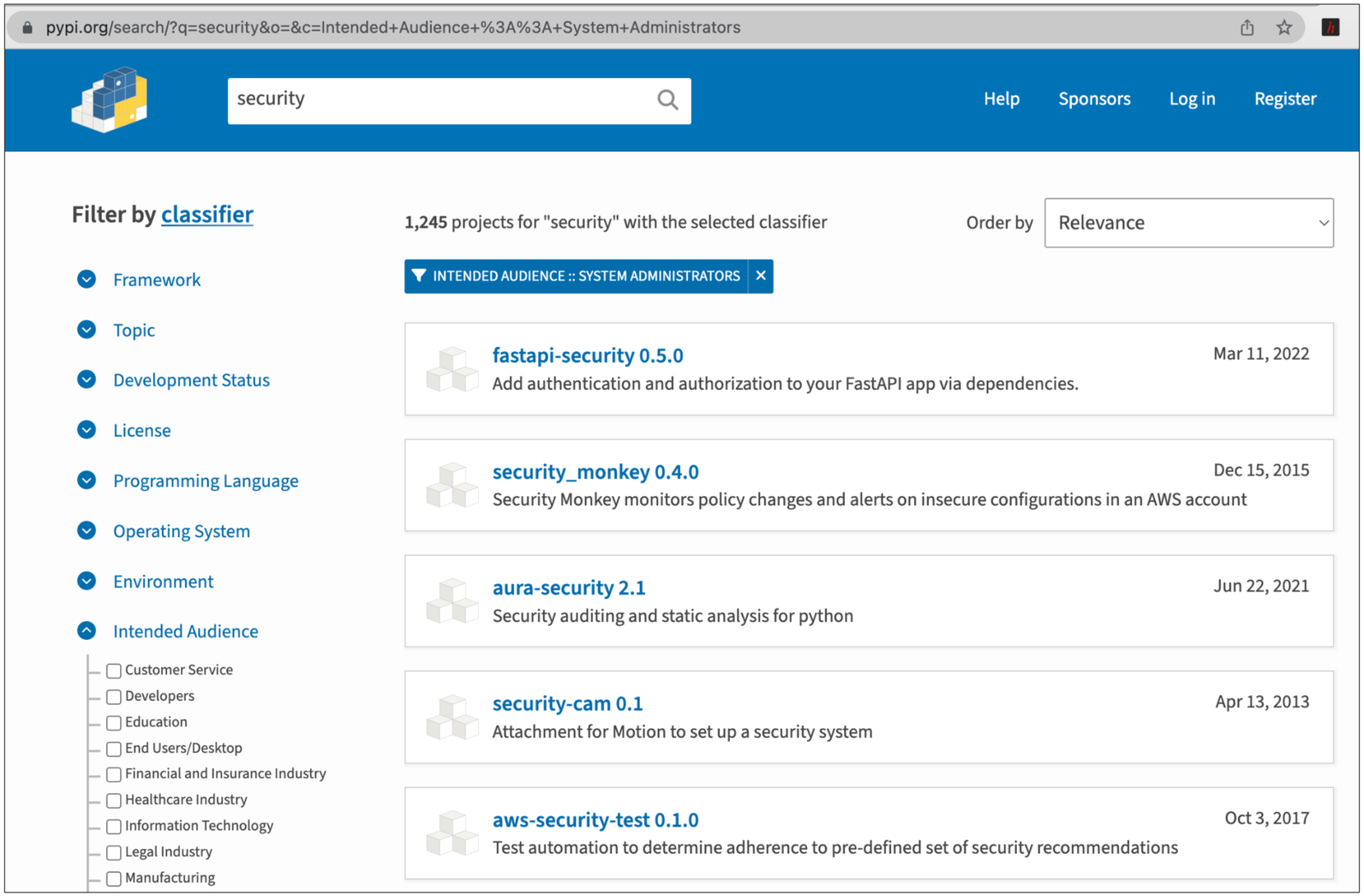


Note: this is bad practice because you don’t always know what you have imported and this can cause naming conflicts, make code harder to read, and clutter the namespace with things you won’t use. Despite this, you will see it used all the time.

Third party Python library

- Third party library = written by someone other than Python developers
- Many useful third party libraries for data science tasks
- Not installed with Python
- Can be installed into the Python environment (pip, conda) or directly in a Jupyter Notebook (pip)
- Download from PyPI (Python Package Index) (pypi.org)
- Imported/used the same way the built-in libraries are imported

Third party Python library



Third party Python library

pypi.org/project/fastapi-security/

Search projects

Help Sponsors Log in Register

fastapi-security 0.5.0

```
pip install fastapi-security
```

Latest version

Released: Mar 11, 2022

Add authentication and authorization to your FastAPI app via dependencies.

Navigation

- Project description
- Release history
- Download files

Project links

- Homepage
- Documentation
- Repository

Statistics

View statistics for this project via Libraries.io, or by using our public dataset on Google BigQuery

Project description

FastAPI Security

ci failing | cd passing | python 3.7 | 3.8 | 3.9 | 3.10 | coverage 97% | pypi package v0.5.0

Add authentication and authorization to your FastAPI app via dependencies.

Installation

With OAuth2/OIDC support:

```
pip install fastapi-security[oauth2]
```

With basic auth only:

```
pip install fastapi-security
```

Pandas: The Series Object

A Series is like a list or a dictionary but BETTER!!

The Series is the primary building block for all the other data structures we will discuss

Python libraries: Common for Data Sci

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Creating a Series

```
import pandas as pd
myData  = pd.Series( <data> )
```

Creating a Series

```
import pandas as pd

series1 = pd.Series( [‘a','b','c','d','e'] )
```

```
series2 = pd.Series( {‘firstName':'Charles',
                      ‘lastName’ : ‘Givre’} )
```

Pandas Series

```
series1 = pd.Series( ['a','b','c','d','e'] )
print(series1)
```

```
0    a
1    b
2    c
3    d
4    e
dtype: object
```

Pandas Series: single elements

```
series1 = pd.Series( ['a','b','c','d','e'] )
print(series1[3])
```

```
d
```

Pandas Series: slicing

```
series1 = pd.Series( ['a','b','c','d','e'] )
print(series1[1:3])
```

```
1     b
2     c
dtype: object
```

Accessing a Series

```
series2 = pd.Series( {'firstName':'Charles',
                      'lastName' : 'Givre'} )

print(series2['firstName'])
```

```
Charles
```

Accessing a Series

```
series2 = pd.Series( {‘firstName':'Charles',
                      ‘lastName’ : ‘Givre’} )

print(series2[[‘firstName’,’lastName’]])
```

```
firstName    Charles
lastName       Givre
dtype: object
```

Accessing a Series

```
series2 = pd.Series( {‘firstName':'Charles',
                      ‘lastName’ : ‘Givre’} )

print(series2[0])
```

```
Charles
```

Accessing a Series

```
randomNumbers = pd.Series(
        np.random.random_integers(1, 100, 50))

randomNumbers.head()
```

```
0     48
1     34
2     84
3     85
4     58
dtype: int64
```

Accessing a Series

```
randomNumbers = pd.Series(
 np.random.random_integers(1, 100, 50) )
randomNumbers.tail( 7 )
```

```
43    66
44    66
45    43
46    55
47    99
48    82
49    19
dtype: int64
```

Pause

Filtering Data in a Series

```
randomNumbers [ <boolean condition> ]
randomNumbers [ randomNumbers < 10 ]
```

```
12     5
21     2
24     1
27     1
29     1
dtype: int64
```

Filtering Data in a Series

```
record.str.contains('Cha')
firstname      True
lastname      False
middle        False
dtype: bool

record[ record.str.contains('Cha') ]
firstname    Charles
dtype: object
```

String Functions

Function	Explanation
Series.str.contains(<pattern>)	Returns true/false if text matches a pattern
Series.str.count(<pattern>)	Returns number of occurrences of a pattern in a string
Series.str.extract(<pattern>)	Returns matching groups from a string
Series.str.find(<string>)	Returns index of first occurrences of a substring (Note: not regex)
Series.str.findall(<pattern>)	Returns all occurrences of a regex
Series.str.len()	Returns the length of text
Series.str.replace(<pat>, <replace>)	Replaces matches with a replacement string

https://pandas.pydata.org/docs/user_guide/text.html#method-summary

Date/Time Operations

```
pd.to_datetime(<times>, format='<format string>')
```

Selected Date/Time Operations

Function	Explanation
`series.dt.year`	Returns the year of the date time
`series.dt.month`	Returns the month of the date time
`series.dt.day`	Returns the days of the date time
`series.dt.weekday`	Returns the day of the week
`series.dt.is_month_start`	Returns true if it is the first day of the month
`series.dt.strftime()`	Returns an array of formatted strings specified by a `date_format`

https://pandas.pydata.org/docs/user_guide/timeseries.html

```
combinedSeries = series1.add( series2 )
```

```
def addTwo(n):
    return n+2

odds.apply(addTwo)
```

```
odds.apply(lambda x: x + 1 )
```

Questions?

In Class Exercise

Worksheet 1.1: Working with One Dimensional Data