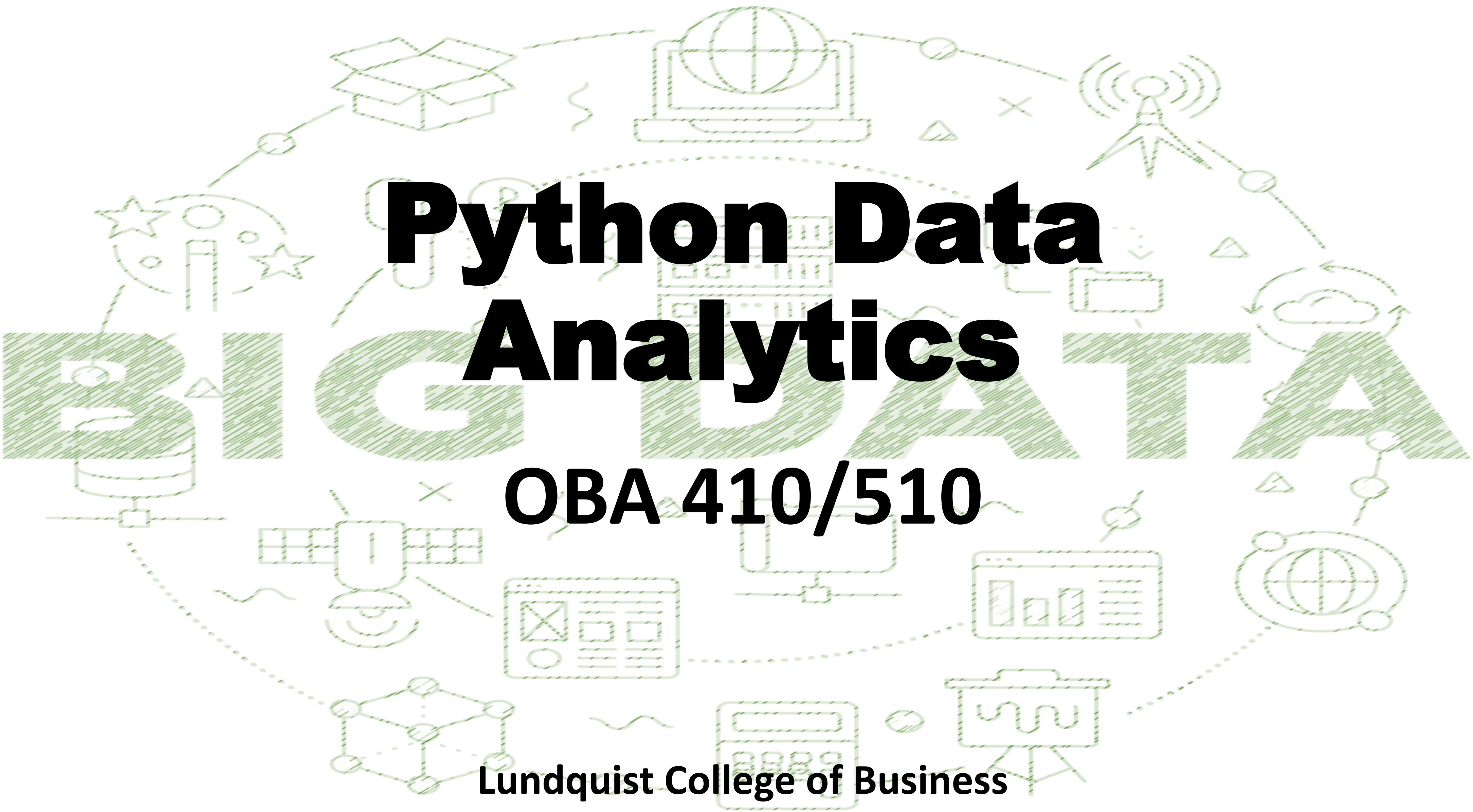

Python Data Analytics

OBA 410/510

Lundquist College of Business

Background image from www. givweb.com

Some of the slides and contents are adopted from Statistical Learning course by Dr. Trevor Hastie

Strings

- Objects that contain sequences of character data

```
In [52]: name="Jessica"

In [53]: name
Out[53]: 'Jessica'

In [54]: address="345 W 13th St"
         address
Out[54]: '345 W 13th St'
```

- We can access individual characters in a string using the letter’s index
  - <u>first character having an index value of **0**</u>

```
In [55]: name[0]
Out[55]: 'J'

In [57]: name[4]
Out[57]: 'i'

In [58]: address[:3]
Out[58]: '345'
```

Methods

- Functions that belong to objects
- For instance *string* is an object
  - Therefore, there are methods for *strings*
- For instance*:*
- *upper()*
  - Return a copy of the string converted to uppercase.
- *count(sub)*
  - Return the number of occurrences of substring sub

```
In [59]: name.upper()
Out[59]: 'JESSICA'

In [63]: address.count('3')
Out[63]: 2
```

Lists

- Lists are a sequence of values that is similar to a string
- Differences with strings:
  - list is a sequence of any type, while a string is only a sequence of characters
  - lists are mutable and we can change elements
- We create lists using square brackets
- Lists can even contain other lists as an element

```
In [66]: list1=[3,5,7,9]
         list1
Out[66]: [3, 5, 7, 9]

In [69]: list2=[3,'hello']
         list2
Out[69]: [3, 'hello']

In [74]: list3=['hi',45,True,[33,1]]
         list3
Out[74]: ['hi', 45, True, [33, 1]]

In [75]: list2[1]='bye'
         list2
Out[75]: [3, 'bye']
```

Slicing and dicing lists

- We use indexes similar to strings

```
In [76]: list1[2:]
Out[76]: [7, 9]

In [77]: list3[3]
Out[77]: [33, 1]

In [78]: list3[2:4]
Out[78]: [True, [33, 1]]

In [80]: list2[1]
Out[80]: 'bye'
```

Adding/removing elements to/from lists

- Listname.append(new elements)
- Listname.remove(elements)

```
list3=[45,54,'Hi',False,54]
```

```
list3.append('Hello')
list3
```

```
[45, 54, 'Hi', False, 54, 'Hello']
```

```
In [15]: list3.remove('Hi')
         list3
Out[15]: [45, 54, False, 54, 'Hello']

In [16]: list3.remove(list3[0])
         list3
Out[16]: [54, False, 54, 'Hello']
```

Defining new Functions-Example

- Defining a function that takes a list of numbers as input and adds up the first and last elements of the list

```
In [20]: def list_add(l1):
             a=l1[0]+l1[-1]
             return a

In [27]: list_add([3,5,7,132])

Out[27]: 135

In [28]: my_list=[3,5,7,132]
         list_add(my_list)

Out[28]: 135
```

Dictionaries

- A dictionary is similar to a list where the indices are not limited to only integers
- The dictionary is a set of key-value pairs where the key is the index to its associated value
- We create dictionaries using curly brackets

```
In [90]: dicionary_1={"a":3,'b':"California", 'c':'Oregon'}

In [92]: dicionary_1["c"]
Out[92]: 'Oregon'

In [87]: dicionary_1['a']
Out[87]: 3

In [93]: dicionary_1['b']='Washington'
         dicionary_1
Out[93]: {'a': 3, 'b': 'Washington', 'c': 'Oregon'}
```

```
In [218]: dicionary_1={"a":3,5:"California", 'c':'Oregon'}

In [220]: dicionary_1[5]
Out[220]: 'California'
```

Python Packages

- Methods and functions are very useful, but they are not enough
- If we do not use packages, we need to write long codes for many simple things
- We do not need to be worried about maintaining every single line in our code, if we use packages
- Packages make our life easier!!
- Before using any package, we have to install and then import it

Python Packages

- scikit-learn
  - Most commonly used library for predictive modeling both in industry and academia
  - It contains most of the machine learning algorithms
  - Comprehensive documentation about each algorithm:
    - http://scikit-learn.org/stable/documentation.html
- NumPy
- SciPy
  - fundamental packages for scientific computing
- Matplotlib
  - Primary scientific plotting library in Python
- Pandas
  - Library for data manipulation and preparation

Python Packages

- NumPy
  - One of the fundamental packages for scientific computing
  - Contains functionalities for <u>multidimensional arrays</u> and linear algebra operations
  - scikit-learn takes in data in the form of Numpy arrays
  - So, any data we use, have to be converted to Numpy arrays
  - The core functionality of Numpy is the *ndarray* class, n-dimensional array
    - *import numpy as np*
    - *x=np.array([[3,5,7],[1,3,9]])*

```
In [4]: x

Out[4]: array([[3, 5, 7],
               [1, 3, 9]])
```

Installing other packages

- You can use *pip* to install packages for Python
- *pip* (PIP Installs Packages or Preferred Installer Program) is a command-line utility **in your console** that allows you to install, reinstall, or uninstall PyPI (Python Package Index) packages
- To install a package use the following command (in your console):
  - *pip install* package-name
- To list all installed packages:
  - *pip list*
- To upgrade an outdated package:
  - *pip install --upgrade* SomeProject
- For more information see the following link:
  - https://packaging.python.org/tutorials/installing-packages/

NumPy

- Start using NumPy by importing it!

```
In [107]: import numpy as np
```

- NumPy *array* is an alternative to Python *list*
- We can do calculations over the entire array
- Operations are very fast
- But NumPy *arrays* can contain only one type of data!
- If you try to create an array with different data types, some of the element's type will change to have a homogeneous array

NumPy arrays

```
In [108]: # Creating one-dimension array
          a1=np.array([4, 8, 1, 0])
          a1
Out[108]: array([4, 8, 1, 0])

In [110]: a1.shape
Out[110]: (4,)

In [111]: #creating two-dimension array
          a2=np.array([[3,5,7],[1,1,1]])
          a2
Out[111]: array([[3, 5, 7],
                 [1, 1, 1]])

In [112]: a2.shape
Out[112]: (2, 3)
```

```
In [118]: # creating an array from a list
          l1=[[11,2,5,7,99],[6,4,4,0,90],[1,0,0,0,32]]
          a4=np.array(l1)
          a4
Out[118]: array([[11,  2,  5,  7, 99],
                 [ 6,  4,  4,  0, 90],
                 [ 1,  0,  0,  0, 32]])
```

```
In [113]: # type coercion
          a3=np.array([4,'Hi', True])
          a3
Out[113]: array(['4', 'Hi', 'True'], dtype='<U11')
```

Max, min, …

- *max()* finds the maximum element in an array
- *min()* finds the minimum element in an array
- *argmax()* finds the index of the maximum element in an array
- *argmin()* finds the index of the minimum element in an array

```
In [140]: a6=np.array([2,6,0,36,14])
          a6
Out[140]: array([ 2,  6,  0, 36, 14])

In [141]: a6.max()
Out[141]: 36

In [142]: a6.argmax()
Out[142]: 3

In [143]: a6.min()
Out[143]: 0

In [144]: a6.argmin()
Out[144]: 2
```

Mean, median, etc.

```
In [33]: list1=[11,4,2,88,0,2,5,-11,2,4]
         list1
Out[33]: [11, 4, 2, 88, 0, 2, 5, -11, 2, 4]

In [25]: np.min(list1)
Out[25]: -11

In [26]: np.argmin(list1)
Out[26]: 7
```

```
In [27]: np.max(list1)
Out[27]: 88

In [28]: np.argmax(list1)
Out[28]: 3

In [29]: np.median(list1)
Out[29]: 3.0

In [31]: np.std(list1)
Out[31]: 26.28706906446589 6
```

pandas

- https://pandas.pydata.org/
- Python library for data cleaning and pre-processing
- It is built around a data structure called DataFrame
- DataFrame is a tabular, column-oriented data structure with both column and row labels
- DtatFrame is a data table similar to an Excel spreadsheet
- panda provides a great range of methods to modify and operate on the structures
- In contrast to NumPy arrays, DataFrame allows each column to have a separate data type (for example integer, float, dates, and string)

panda's Data Structures

- **Series**
  - One-dimensional ndarray with axis labels
  - A cross between a list and a dictionary
  - Items are all stored in an order and there's labels with which you can retrieve them
  - We can create Series from
    - *lists* (the index will be incrementing integers),
    - *dictionaries* (the index is automatically assigned from the keys of the dictionary),
    - Or from scratch by passing a list of data, and index to the series (If you don't give an index to the series, the index will be incrementing integers)

Series

```
In [18]: import pandas as pd

In [32]: sports=['football','soccer','Taekwondo']

In [33]: s1=pd.Series(sports)
         s1

Out[33]: 0     football
         1       soccer
         2    Taekwondo
         dtype: object
```

```
In [36]: sports_dic={'US':'football','Argentina':'soccer','South Korea':'Taekwondo'}

In [38]: s2=pd.Series(sports_dic)
         s2

Out[38]: US              football
         Argentina         soccer
         South Korea    Taekwondo
         dtype: object
```

```
In [39]: s3 = pd.Series(['football','soccer','Taekwondo'], index=['US','Argentina','South Korea'])
         s3

Out[39]: US              football
         Argentina         soccer
         South Korea    Taekwondo
         dtype: object
```

Querying Series

- A panda Series can be queried, either by the index position or the index label

- To query by numeric location, starting at zero, use the ***iloc*** attribute. To query by the index label, you can use the ***loc*** attribute.

- Keep in mind that *iloc* and *loc* are not methods, they are attributes. So you don't use parentheses to query them, but square brackets instead, which we'll call the indexing operator.

```
In [40]: s3.iloc[0]
Out[40]: 'football'

In [43]: s3.loc['US']
Out[43]: 'football'

In [42]: s3.loc['Argentina']
Out[42]: 'soccer'
```

Modify/Adding New Data

- *loc* attribute lets you modify data in place and also add new data
- If the value you pass in as the index doesn't exist, then a new entry is added
- Indices can have mixed types

```
In [57]: s3.loc['US']='Baseball'
         s3

Out[57]: US               Baseball
         Argentina          soccer
         South Korea     Taekwondo
         dtype: object

In [58]: s3.loc['Canada']='hockey'
         s3

Out[58]: US               Baseball
         Argentina          soccer
         South Korea     Taekwondo
         Canada             hockey
         dtype: object
```

Deleting data from Series

- Series.drop(labels=None, inplace=False)
  - labels : single label or list-like
    - Index or column labels to drop.
  - inplace : bool, default False
    - If True, do operation inplace and return None.

```
In [9]:  s3 = pd.Series(['football','soccer','Taekwondo'], index=['US','Argentina','South Korea'])
         s3

Out[9]:  US              football
         Argentina         soccer
         South Korea    Taekwondo
         dtype: object

In [10]: s3.drop('US')

Out[10]: Argentina         soccer
         South Korea    Taekwondo
         dtype: object
```

panda’s Data Structures

- **DataFrame**
  - The *DataFrame* data structure is the heart of the panda's library
  - The *DataFrame* is conceptually a two-dimensional *Series* object
  - There is an index and multiple columns of content, with each column having a label

```
In [10]: pd.DataFrame([['Milk','Bread','Beer'],[12,8,25]]).T
```

Out[10]:

	0	1
0	Milk	12
1	Bread	8
2	Beer	25

```
In [64]: df = pd.DataFrame(data={'Product':['Milk','Bread','Beer'],'Cost':[12, 8 ,25]})
         df
```

Out[64]:

	Product	Cost
0	Milk	12
1	Bread	8
2	Beer	25

```
df2=pd.DataFrame(data={'Product':['Milk','Bread','Beer'],'Cost':[12,8,25]},
                 index=['Trans1','Trans2','Trans3'])
df2
```

	Product	Cost
Trans1	Milk	12
Trans2	Bread	8
Trans3	Beer	25