Numpy and Pandas

Outcomes

1. Numpy arrays
    1. Kinds of data
    2. Why arrays vs lists
    3. Creating arrays
    4. Array attributes
    5. Operations on Arrays
    6. Array slicing, reshaping, transposing
2. Pandas
    1. Series
    2. Dataframes

Data

https://jakevdp.github.io/PythonDataScienceHandbook/02.00-introduction-to-numpy.html

- Datasets come from a wide range of sources in a wide range of formats
- collections of documents, collections of images, collections of sound clips, collections of numerical measurements, or nearly anything else.
- Fundamentally, we think of it all as ***arrays of numbers***

Numpy

- efficient storage and manipulation of numerical arrays is absolutely fundamental to the process of doing data science.
- NumPy arrays form the core of nearly the entire ecosystem of data science tools in Python
- time spent learning to use NumPy effectively will be valuable no matter what aspect of data science interests you.

Numpy

- 7.2 creating arrays from lists and multi-dimentional lists
- 7.3 array attributes: *array_object.attribute*
    - *dtype*
    - ndim
    - shape
    - itemsize
    - flat
-

7.5 creation/initialization

- Functions take integer or tuple (for multidimentional) argument
- *np.full()* Also needs a value argument
- *np.zeros()* (default float64)
- *np.ones()* (default float64)
- Can also take a *dtype* argument
- *np.arange*() like *range()* but better for arrays
- *np.linspace(first,last[,num])* num defaults to 50

7.5 array object reshape function

7.5 displays large arrays with ,…, for large numbers of rows and/or columns

Time to check your learning!

Let’s see how many key concepts from Numpy you recall by answering the following questions!

- What's the fundamental difference between a Numpy array and a python list?
- How can we create an array from a list?
- How can we find the datatype of the elements of an array?
- How can we transform an array into a different shape?
- How can we create an array with evenly spaced floating point values?

7.6 IPython %timeit magic command

7.6 np.random.randint() function to generate random-valued array

7.6 other potential IPython magics

7.7 Array operators

- Can perform element-wise operations on arrays of the same shape (and we'll see later also different shapes)
- With a scalar, the value is *broadcast*, as if it were an array of the same shape
- Comparison operators give arrays of boolean values
- 7.8 array calculations
    - Functions *sum, min, max, mean, std, var*

7.8 calculations by row or column

- specify an integer *axis* keyword argument to indicate dimension

- https://docs.scipy.org/doc/numpy/reference/arrays.ndarray.html
- 7.9 Universal functions
  - *np.function(array)* as opposed to *array.min()* on previous slide
- 7.9 Broadcasting with universal functions
  - Example two-dimensional array multiplied by a row
- 7.9 Many universal functions
  - https://docs.scipy.org/doc/numpy/reference/ufuncs.html

- 7.10 Indexing and slicing
  - Two-dimensional indexing: *array[x,y]*
  - To retrieve a row from two-dimensional: *array[x]*
  - Two-dimensional slicing on rows: *array[first:last]*
    - *last* is non-inclusive, as before
  - Select a list of rows: *array[[1,3,4]]*
  - Can also slice columns: *array[:,first:last]*
    - The first *:* selects all rows, the second *:* selects the columns
  - Combine any of the above: *array[fr:lr,fc:lc]*
    - Selects the sliced rows and columns

- 7.11 Views: Shallow Copies
    - Create a complete view with *array.view()*
    - Slices create views
- 7.12 Deep Copies
    - Create a deep copy with *array.copy()*
- 7.13 Reshaping and Transposing
    - *array.reshape()* and *array.resize()* both change the dimensions of an array
        - reshape returns a new view of unmodified array
        - resize actually modifies the array

- 7.13 Reshaping and Transposing
  - You can take a multi-dimensional array and obtain a one-dimensional array with *flatten* and *ravel*
    - *array.flatten()* returns a deep copy of (unmodified) array
    - *array.ravel()* returns a view of (unmodified) array
  - Transposing
    - The *array.T* attribute returns a transposed view of array
  - Horizontal and Vertical Stacking
    - Pass a tuple of arrays to *numpy.hstack()* or *numpy.vstack()*
    - *numpy.hstack((array1, array2))* returns longer rows
    - *numpy.vstack((array1,array2))* returns more rows

Time to check your learning!

Let’s see how many key concepts from Numpy you recall by answering the following questions!

- What does *Broadcasting* mean with numpy arrays?
- How can we find the average of all the values in an array?
- How can we find the average value of all the rows of an array?
- How can we find the average value of all the columns of an array?
- How can we extract the first 3 columns of the first 3 rows?
- How can we find the transposition of a 2-dimensional array?

Why Pandas?

- https://towardsdatascience.com/a-python-pandas-introduction-to-excel-users-1696d65604f6
- Row and column labels (DataFrames)
- Missing data
- Heterogeneous data types
- powerful data operations like those offered by database engines and spreadsheets
- Efficiency when dealing with large amounts of data

Pandas

- https://pandas.pydata.org/pandas-docs/stable/reference/
- Pandas Series: one dimensional
- Pandas DataFrame: two dimensional
- Closely related to Numpy arrays
- 7.14.1 Creating Series
    - Default indices: *pandas.Series([45,7,23])*
    - Displayed as two columns: indices and values
    - All same value 80, supply indices: *pandas.Series(80,[45,7,23])*
    - *index* is a keyword argument: *pandas.Series(80,index=[45,7,23])*
    - Any iterable can be used as indices: *pandas.Series(80,range(3))*
    - Create with a dictionary, and the dictionary keys become indices

- 7.14.1 accessing elements by index: *seriesname[index]*
  - If indices are valid python identifiers (strings), can access values as if they were attributes: *seriesname.index*
  - Access all values: *seriesname.values* note *values* is attribute
  - Access all keys: *seriesname.keys()*
- 7.14.1 computing stats: *count, min, max, mean, var, std*
  - Example *seriesname.count()*
- 7.14.1 quartiles and all the above produced by *seriesname.describe()*
  - *25%*: median of first half of sorted values
  - *50%*: median of all sorted values
  - *75%*: median of second half of sorted values

- 7.14.1 if *seriesname* contains strings can use the *str* attribute to call string methods on the elements
    - *seriesname.str.contains('a')* returns a new series with values True or False corresponding to each element
    - *seriesname.str.upper()* returns a new series with allcaps values

- 7.14.2 DataFrames
  - Enhanced 2-d arrays
  - Can deal with missing data
  - Each column is a Series
  - Different columns can contain a different data types
- 7.14.2 Creating a DataFrame from a dictionary
  - Either values must be lists or you must supply index parameter
  - *pd.DataFrame({'me':[0],'him':[10],'her':[20]})*
  - *me, him, her* are the column names
  - *[0], [10], [20]* are the (one-row) columns
  - Indices aren't specified so they are default 0, 1, 2

- 7.14.2 Accessing columns by name results in a Series
  - *dataframename[columnname]* or
  - *dataframename.columnname* if columnname is valid identifier
- 7.14.2 Recommended to use *loc, iloc, at, iat* attributes for access:
  - *dataframename.loc['label']* gives the row with explicit *label*
  - *dataframename.iloc[int]* gives the row at position *int* (0-based)
  - Can use slices:
    - *dataframename.loc['label1':'label2']* **includes** label2!
    - *dataframename.iloc[int1:int2]* **does not include** int2
  - Can use specific lists of individual rows:
    - *dataframename.loc[['label1','label2']]*
    - *dataframename.iloc[[int1,int2]]*

- 7.14.2 Can specify subsets of columns by including a second slice or list:
  - *dataframename.loc['row1':'row2',['col1','col3']]*
  - *dataframename.loc[['row1','row2'],['col1','col3']]*
  - *dataframename.loc[['row1','row2'],'col1':'col3']*
  - *dataframename.loc['row1':'row2','col1':'col3']*
- 7.14.2 Similarly for *iloc*
- 7.14.2 Boolean indexing
  - *dataframename[dataframename > 90]* evaluates to a dataframe with elements <= 90 changed to NaN
  - Can combine conditions with & and |

- 7.14.2 Accessing a single cell by row and column
  - Use *at* and *iat* attributes to access (including change) specific cell values
  - Two string values (at) or two integer values (iat) separated by comma
  - *dataframename.at['rowlabel','columnlabel']*
  - *dataframename.iat[rowindex,columnindex]*
- 7.14.2 Descriptive statistics
  - *describe()* method for both Series and DataFrames
  - Statistics calculated by column
  - Returned in the form of a DataFrame

- 7.14.2 Controlling Pandas precision
  - Use the set_option function: *pd.set_option('precision', 2)*
  - Default is 6, above would set 2 digits after decimal point
- 7.14.2 *dataframename.mean()* gives mean of each column
- 7.14.2 *dataframename.T* is a transposed view
- 7.14.2 *dataframename.T.mean() gives mean of each row*
- 7.14.2 Sorting DataFrames by row and column