Lecture 2

Data Wrangling with NumPy & Pandas

July 30

NumPy - Introduction

- NumPy, short for Numerical Python.
- fundamental package for high performance scientific computing and data analysis.

- Provides
  - Standard mathematical functions for fast vectorized array operations.
  - Efficient descriptive statistics and summarizing data.
  - Tools for reading / writing array data to disk.
  - Linear algebra, random number generation, and Fourier transform capabilities
- How to import?
  - Import numpy as np

NumPy - ndarrays

- A Multidimensional Array Object.
- Generic multidimensional storage for homogeneous data.
- How to define an array? Multiple ways.
  - data = np.random.rand(d0, d1) # create a random array with dimensions <d0, d1>
  - data = np.random.randn(d0, d1) # create a random array from normal distribution.
  - data = np.arange(start, end) # create a sequence of integers from <start> to <end>
  - data = np.array(input_data) # creates an ndarray from list, tuple, array, etc.
  - data = np.ones(d0, d1) # creates an array of 1's. Try *np.zeros(), np.eye()*.
- Has a **shape.** How to find it?
  - data.shape
- How to find the type of data being stored?
  - data.dtype

ndarray - data types

- data.dtype
  - Output: dtype(‘float64’)
- Dtypes for ndarray are made up of 2 parts:
  - a type name (int, float)
  - Number; indicating the ‘bits per element’.

- Double precision float: 64 bits. Other frequently used dtypes:
  - float32 : single precision floating point (Standard)
  - float16 : half precision floating point.
  - int16 : signed 16-bit integer types.
  - bool : Boolean type (True/False)

ndarray - Arithmetic operations

- ndarrays - allows batch operations on data without for loops.
- Arithmetic operation on equal sized arrays - always element-wise
  - data = np.random.randn(2,4)
  - data*data # element wise product operation
  - data - data # difference operation. Solution?
  - 1/data # 1/element
  - data**0.5 # square root operation

- Operations between different sized arrays - Broadcasting.

Ndarray - Indexing & Slicing - 1D

- Many ways to select a subset of the data.
  - data = np.arange(10)
  - data[5] # select the element at 6th index. Remember first index starts is at 0.
  - data[5:8] # select from 5th to 7th element. Stop index not included.
  - data[5:8] = 6
  - data[:] = 0 # ‘bare’ slice refers to all elements

- NumPy: provides a view rather than a copy for slicing
  - For explicit copy: data[5:8].copy()

Ndarray - Indexing & Slicing - Higher dimension

- Indexing
  - data_2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
  - data_2d[2] # selects the 3rd row
  - data_2d[0][2] # select elements with <row, column> indexing format
  - data_2d[0, 2] # another way for indexing

- Slicing: select range of elements
  - data_2d[:2] # slicing along axis 0 (row): select everything above row#3
  - data_2d[1, :2] # practice in colab

ndarray - Transposing & Swapping Axes

- Transposing: special form of reshaping.
  - data = np.arange(15).reshape((3, 5))
  - data.shape # check in colab
  - data.T # transpose of the original ndarray
- For high dimension
  - data_3d = np.arange(16).reshape((2, 2, 4)) # accepts a tuple of axis
  - data_3d.transpose((1, 0, 2))

- Are you creating a copy or view?
- Check what .T will do on high dimensional data?
- Try data_3d.swapaxes(1,2)

ndarray - Statistical methods

- Mathematical functions to compute statistics about an entire array.
- Also called Aggregation (sum, mean, std)
  - data = np.random.randn(5, 4) # normally-distributed data
  - data.mean() # Average of the array
  - np.mean(data) # same operation
- For high dimension data
  - data_2d = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8]])
  - data_2d.mean(axis=1) # mean over the given axis
  - data_2d.sum(0) # check in colab

ndarray - Unique

- For 1-D ndarrays.
- returns the sorted unique values in an array

- data_ints = np.array([3, 3, 3, 2, 2, 1, 1, 4, 4])
  - np.unique(data_ints) # Output: array([1, 2, 3, 4])

Pandas - Introduction

- Contains tools for data structure & manipulation.
- Has good support for NumPy, SciPy, statsmodel, scikit-learn, matplotlib, etc.
- Designed to work with tabular & heterogeneous data.
  - Numpy good for homogeneous data.

- How to import?
  - import pandas as pd

Pandas - Data Structures

- 2 main data structures
  - Series
  - DataFrame
- Series: 1-D array like object
  - Contains sequence of homogeneous data.
  - ser = pd.Series([4, 5, 10, 3])
  - ser # execute in colab & spot difference from ndarray
  - ser.values
  - ser.index
- Filtering
  - ser[ser>6] # does it affect the index?
  - ser *2

Pandas - DataFrame

- For tabular, spreadsheet-like data structure.
  - contains an ordered collection of columns
    - each of which can be a different value type (numeric, string, boolean, etc.)
- Has both row & column index

- How to create?
  - A popular way through dict of equal length list or ndarrays
    - data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'], 'year': [2000, 2001, 2002, 2001, 2002], 'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
    - df = pd.DataFrame(data)

Indexing, Selection, Index dropping

- Series indexing
  - obj = Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
  - obj[‘b’]
  - Obj[1]

- Dropping one or more entries from an axis
  - obj = Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
  - new_obj = obj.drop(‘c’)
  - obj.drop([‘d’, ‘c’])

Indexing, Selection, Index dropping -2

- Selection with loc & iloc
  - Select rows & columns using axis labels (loc) or integers (iloc).

  - data = DataFrame(np.arange(16).reshape((4, 4)),index=['Ohio', 'Colorado', 'Utah', 'New York'], columns=['one', 'two', 'three', 'four'])

  - data.loc[‘Colorado’, [‘two’, ‘three’]]
  - data.iloc[2, [3, 0, 1]]

Pandas - Descriptive Statistics

- Dataframes & Series objects are equipped with a set of common mathematical and statistical methods
  - df = DataFrame([[1.4, np.nan], [7.1, -4.5], [np.nan, np.nan], [0.75, -1.3]], index=['a', 'b', 'c', 'd'], columns=['one', 'two'])
  - df.sum()
  - df.head()
  - df.tail()
  - df.summary()

Colab Notebook

Colab Notebook for Lec 2