Numpy and pandas Basics

¶

Neba Nfonsang
University of Denver

In [1]:
```
# import necesssary packages
import numpy as np
import pandas as pd
```

NumPy Basics

Vectors and Matrices

NumPy's one-dimensional arrays are vectors while NumPy's two-dimensional arrays are matrices.

Vectors

In [2]:
```
# create a one-dimensional array or a vector (as a row)
row_vector = np.array([1, 2, 3, 4])
row_vector
```

Out[2]:
```
array([1, 2, 3, 4])
```

In [3]:
```
# create a one-dimensional array or a vector (as a column)
column_vector = np.array([[1], [2], [3], [4]])
column_vector
```

Out[3]:
```
array([[1],
       [2],
       [3],
       [4]])
```

In [4]:
```
# create a vector using np.arange(start, stop, step)
vector = np.arange(1, 11, 2)
vector
```

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

In [5]:
```
# create a vector with np.zeros()
zero_vector = np.zeros(5)
zero_vector
```

Out[5]:
```
array([0., 0., 0., 0., 0.])
```

In [6]:
```
# create a vector with np.ones()
ones_vector = np.ones(5)
ones_vector
```

Out[6]:
```
array([1., 1., 1., 1., 1.])
```

Matrices

In [7]:
```
# create a matrix with a nested list of lists
nested_list = [[1,2,3], [4, 5, 6], [7, 8, 9]]
matrix = np.array(nested_list)
matrix
```

Out[7]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

In [8]:
```
# create an identity matrix with np.identity()
identity_matrix = np.identity(5)
identity_matrix
```

Out[8]:
```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

Random Number Generation

Use features inside the the np.random module to generate random numbers or samples from specific distributions. This module also contains functions for simulating various distributions such as the normal, uniform, binomial, beta, and gama distributions.

use the syntax **?np.random** to view the functions inside the np.random module.

In [9]:
```
# generate uniformly distributed values between 0 and 1
# one sample (a vector of value)

np.random.rand(5)
```

Out[9]:
```
array([0.35902915, 0.33973831, 0.05369295, 0.61729455, 0.05463598])
```

In [10]:
```
# generate uniformly distributed values between 0 and 1
# 5 samples (a matrix)

np.random.rand(5, 3)
```

Out[10]:
```
array([[0.87351474, 0.32287261, 0.71197115],
       [0.10744304, 0.49118211, 0.88961779],
       [0.41300936, 0.1391197 , 0.92168114],
       [0.77427761, 0.43734382, 0.13392538],
       [0.33151838, 0.80937616, 0.78592352]])
```

In [11]:
```
# generate values from the standard normal distribution (z)
# one sample

np.random.randn(5)
```

Out[11]:
```
array([ 0.03285519, -1.76232967, -0.20227057, -0.49058212, -0.62160759])
```

In [12]:
```
# generate values from the standard normal distribution (z)
# 5 samples (a matrix)

np.random.randn(5, 4)
```

Out[12]:
```
array([[ 0.08466412,  1.16962923,  0.810379  ,  0.1662918 ],
       [ 0.69954203,  0.92570213, -0.34388084,  1.28002395],
       [-1.7217007 , -0.25145949,  1.29088066,  0.565541  ],
       [-0.45507599, -0.20472464, -0.55738184, -2.49202631],
       [-0.78271432, -0.38759073,  1.33329539,  0.33246713]])
```

In [13]:
```
# generate a random integer from a discrete uniform distribution
# generate a value between 3 and 10

np.random.randint(3,10)
```

Out[13]:
```
9
```

In [14]:
```
# generate a sample of integers between 3 and 10
# with a sample size of 5 from a discrete uniform distribution

np.random.randint(3, 10, size=5)
```

Out[14]:
```
array([9, 4, 6, 3, 3])
```

In [15]:
```
# generate a sample of uniformly distributed floats over [0, 1)
np.random.random_sample(5)
```

Out[15]:
```
array([0.01730982, 0.31970834, 0.91186439, 0.8060887 , 0.02943624])
```

In [16]:
```
# select a random sample of items from a population
population = ["A", "B", "C", "D", "E", "F,", "G"]

sample = np.random.choice(population, size=3, replace=False)
sample
```

Out[16]:
```
array(['A', 'E', 'F,'], dtype='<U2')
```

Generate Probability Distributions

In [17]:
```
# generate a normal distribution
# with a mean of 10 and standard deviation of 3

np.random.normal(loc=120, scale=3, size=10)
```

Out[17]:
```
array([115.36448814, 127.56985086, 124.73891148, 121.37375271,
       119.23923408, 119.83852928, 120.96242955, 113.13817808,
       119.1466134 , 122.75185997])
```

In [18]:
```
# generate a uniform distribution
# with a minimum value of 5 and maximum value of 20

np.random.uniform(low=5, high=20, size=10)
```

Out[18]:
```
array([11.13978942, 10.62296295, 14.56779177, 17.95648441,  6.17568924,
       12.42669886,  6.79043328, 12.9130045 , 13.07911644, 13.15040128])
```

In [19]:
```
# generate a binomial distribution
# with 0.2 as probability of success per trial
# and 10 as total number of trials

np.random.binomial(n=10, p=0.2, size=10)
```

Out[19]:
```
array([1, 1, 3, 2, 3, 2, 1, 2, 1, 5])
```

In [20]:
```
# a and b keeps track of number of successes
# and failures respectively in a beta distribution

np.random.beta(a=5, b=9, size=10)
```

Out[20]:
```
array([0.6316174 , 0.37392   , 0.14814652, 0.52705841, 0.42072436,
       0.4299106 , 0.44841164, 0.21544049, 0.30542437, 0.19317644])
```

In [21]:
```
# set a random seed
np.random.seed(1)

vector = np.random.randint(1, 5, size=12)
vector
```

Out[21]:
```
array([2, 4, 1, 1, 4, 2, 4, 2, 4, 1, 1, 2])
```

In [22]:
```
# reshape the vector of values to a matrix
matrix_2 = vector.reshape(3, 4)
matrix_2
```

Out[22]:
```
array([[2, 4, 1, 1],
       [4, 2, 4, 2],
       [4, 1, 1, 2]])
```

Operations with NumPy Arrays

In [23]:
```
# element wise arithmetic operations
items = ["orange", "apple", "mango", "lemon"]
quantity = np.array([20, 30, 15, 25])
unit_cost = np.array([0.95, 2.00, 1.5, 0.90])
unit_price = np.array([1.25, 2.15, 1.75, 1.30])
```

In [24]:
```
# compute total cost of each item
total_cost = quantity*unit_cost
total_cost
```

Out[24]:
```
array([19. , 60. , 22.5, 22.5])
```

In [25]:
```
# compute revenue assuming all items are sold
revenue = quantity*unit_price
revenue
```

Out[25]:
```
array([25.  , 64.5 , 26.25, 32.5 ])
```

In [26]:
```
# compute profit for the items
profit = revenue - total_cost
profit
```

Out[26]:
```
array([ 6.  ,  4.5 ,  3.75, 10.  ])
```

Indexing Vectors

In [27]:
```
# create a vector
my_vector = np.array([2, 5, 7, 8, 10])
my_vector
```

Out[27]:
```
array([ 2,  5,  7,  8, 10])
```

In [28]:
```
# extract the first element of the vector
# remember Python starts counting from zero (0)

my_vector[0]
```

Out[28]:
```
2
```

In [29]:
```
# extract the third element of the vector
my_vector[2]
```

Out[29]:
```
7
```

In [30]:
```
# extract from the second to the fourth element
my_vector[1:4]
```

Out[30]:
```
array([5, 7, 8])
```

In [31]:
```
# extract the last element using negative indexing
my_vector[-1]
```

Out[31]:
```
10
```

Indexing and Slicing Matrices

In [32]:
```
# create a matrix
my_matrix = np.array([[2, 4, 6, 8], [1, 2, 3, 4], [1, 3, 5, 7]])
my_matrix
```

Out[32]:
```
array([[2, 4, 6, 8],
       [1, 2, 3, 4],
       [1, 3, 5, 7]])
```

In [33]:
```
# slice the first row
my_matrix[0]
```

Out[33]:
```
array([2, 4, 6, 8])
```

In [34]:
```
# slice the second row
my_matrix[1]
```

Out[34]:
```
array([1, 2, 3, 4])
```

In [35]:
```
# slice the first column
my_matrix[:, 0]
```

Out[35]:
```
array([2, 1, 1])
```

In [36]:
```
# alternative way of slicing the first column
my_matrix[0:3, 0]
```

Out[36]:
```
array([2, 1, 1])
```

In [37]:
```
# slice from the first column through the second column
my_matrix[:, 0:2]
```

Out[37]:
```
array([[2, 4],
       [1, 2],
       [1, 3]])
```

In [38]:
```
# extract the value at the intersection
# of the first row and second column

my_matrix[0][1]
```

Out[38]:
```
4
```

Boolean or Conditional Slicing

In [39]:
```
my_matrix
```

Out[39]:
```
array([[2, 4, 6, 8],
       [1, 2, 3, 4],
       [1, 3, 5, 7]])
```

In [40]:
```
# return only entries greater than 4
my_matrix[my_matrix>4]
```

Out[40]:
```
array([6, 8, 5, 7])
```

Let's say the first, second, and third rows of my_matrix correspond to the records or observations of John, Mary, and Peter

In [41]:
```
# create an array of names
names = np.array(["John", "Mary", "Peter"])
names
```

Out[41]:
```
array(['John', 'Mary', 'Peter'], dtype='<U5')
```

In [42]:
```
# select John's record
my_matrix[(names=="John")]
```

Out[42]:
```
array([[2, 4, 6, 8]])
```

In [43]:
```
# select all records except John's record
my_matrix[~(names=="John")]
```

Out[43]:
```
array([[1, 2, 3, 4],
       [1, 3, 5, 7]])
```

In [44]:
```
# select the records of John and Peter without using ~
my_matrix[(names=="John")|(names=="Peter")]
```

Out[44]:
```
array([[2, 4, 6, 8],
       [1, 3, 5, 7]])
```

Attributes of the NumPy Array

In [45]:
```
my_matrix
```

Out[45]:
```
array([[2, 4, 6, 8],
       [1, 2, 3, 4],
       [1, 3, 5, 7]])
```

In [46]:
```
# check the data type of my_matrix array
my_matrix.dtype
```

Out[46]:
```
dtype('int32')
```

In [47]:
```
# check the shape of the array
my_matrix.shape
```

Out[47]:
```
(3, 4)
```

In [48]:
```
# check the dimensions of the array
my_matrix.ndim
```

Out[48]:
```
2
```

In [49]:
```
# check the length of the array
# len() is a Python function

len(my_matrix)
```

Out[49]:
```
3
```

Pandas Data Structure

pandas is a Python library mostly used for data preparation, cleaning and manipulation. Pandas has two fundamental data structures called **Series** and **DataFrame** . Use the syntax **import pandas as pd** to import the pandas library

pandas Series

A pandas Series is a one-dimensional, array-like object containing a sequence of values with associated index or row labels

In [50]:
```
# create a pandas Series
series_1 = pd.Series([2, 4, 6, 8])
series_1
```

Out[50]:
```
0    2
1    4
2    6
3    8
dtype: int64
```

In [51]:
```
# create a pandas Series
# use ["a", "b", "c", "d"] as row labels

series_2 = pd.Series([2, 4, 6, 8], index = ["a", "b", "c", "d"])
series_2
```

Out[51]:
```
a    2
b    4
c    6
d    8
dtype: int64
```

In [52]:
```
# call the dir() function on a Series object
# to see various Series attributes
```

pandas DataFrame

A pandas DataFrame is a two-dimensional data structure that holds homogeneous or heterogeneous data in a tabular or spreadsheet-like format.

In [53]:
```
# create a pandas DataFrame using a zipped list

name = ["John", "Mary", "Rob", "Jane"]
age = [25, 26, 27, 22]
city = ["Denver", "Aurora", "Aurora", "Denver"]

zipped_data = list(zip(name, age, city))
data = pd.DataFrame(zipped_data, columns=["name", "age", "city"])
data
```

Out[53]:

	name	age	city
0	John	25	Denver
1	Mary	26	Aurora
2	Rob	27	Aurora
3	Jane	22	Denver

In [54]:
```
# create a pandas DataFrame using a dictionary

my_dictionary = {"name": ["John", "Mary", "Rob", "Jane"],
                 "favorite_animal": ["cat", "dog", "dog", "fish"]}

fav_animal_data = pd.DataFrame(my_dictionary)
fav_animal_data
```

Out[54]:

	favorite_animal	name
0	cat	John
1	dog	Mary
2	dog	Rob
3	fish	Jane

In [55]:
```
# create a DataFrame using an ndarray
my_array = np.random.randint(low=70, high=100, size=20).reshape(5,4)
score_data = pd.DataFrame(my_array, columns=list("ABCD"))
score_data
```

Out[55]:

	A	B	C	D
0	82	77	83	98
1	76	95	88	90
2	75	88	90	81
3	98	80	98	99
4	84	88	74	93

Attributes of a pandas DataFrame Object

Call the dir() function on a DataFrame object to view the attributes of a DataFrame

In [56]:
```
# display previously created DataFrame
data
```

Out[56]:

	name	age	city
0	John	25	Denver
1	Mary	26	Aurora
2	Rob	27	Aurora
3	Jane	22	Denver

In [57]:
```
# display the shape of DataFrame object
data.shape
```

Out[57]:
```
(4, 3)
```

In [58]:
```
# display the dimensions of the DataFrame object
data.ndim
```

Out[58]:
```
2
```

In [59]:
```
# display the length of the DataFrame object
len(data)
```

Out[59]:
```
4
```