BETHLEHEM UNIVERSITY
جامعة بيت لحم

JÜLICH
Forschungszentrum

Introduction to Python: Basic Python, Pandas, Matplotlib

Anas Samara
asamara@bethlehem.edu

2023, AUGUST 17

Outline

- Package Installer for Python
- Numpy
- Pandas
  - Series
  - Dataframe
  - Loading and manipulating dataset
- HIFIS - The Pandas Framework workshop

Package Installer for Python (PIP)

- PIP is the Package Installer for Python
- You can use pip to install packages from the Python Package Index
- PIP is included by default in Python version 3.4 or later

- Today we are going to use pip-tool to install Numpy and Pandas libraries
  - **`> pip install numpy`**
  - **`> pip install pandas`**

Numpy

- NumPy is a Python library created in 2005
- It is generally used for working with arrays.
- Used for high performance computing and data analysis
  - Internally stores data in a contiguous block of memory, independent of other built-in Python objects
  - Use much less memory than built-in Python sequences.
  - Standard math functions for fast operations on entire arrays of data without having to write loops
  - NumPy Arrays are important because they enable you to express batch operations on data without writing any *for* loops.

ndarray

- Every array must have a **shape** and a **dtype**

```
import numpy as np
data1 = [6, 7.5, 8, 0, 1]
arr1 =  np.array(data1)
print(arr1)         # [6.  7.5 8.  0.  1. ]
print(arr1.dtype) # float64
print(arr1.shape) # (5,)
print(arr1.ndim)  # 1
```

Creating and initializing arrays using numpy

```
data2 = [[1, 2, 3, 4], [5, 6, 7, 8]]
arr2 = np.array(data2)
print(arr2.ndim)   # 2
print(arr2.shape)  # (2, 4)
```

```
np.array([[0,1,2],[2,3,4]])
```

```
[[0 1 2]
 [2 3 4]]
```

```
np.random.randint(50, 100, (3,3))
```

```
[[59 63 59]
 [90 86 85]
 [53 86 54]]
```

```
np.ones((2,3))
```

```
[[1. 1. 1.]
 [1. 1. 1.]]
```

```
np.zeros((2,3))
```

```
[[0. 0. 0.]
 [0. 0. 0.]]
```

Creating and initializing arrays using numpy

```
np.arange(0, 10)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

```
np.arange(0, 21, 5)
```

```
[ 0  5 10 15 20]
```

```
# Identity matrix
np.eye(4)
```

```
[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]
```

```
np.eye(4, k=1)
```

```
[[0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]
 [0. 0. 0. 0.]]
```

Element-wise Matrix arithmetic

```
# Element-wise matrix multiplication
ar1 = np.array([[1,2,3],[1,2,3]])
ar2 = np.array([[1,2,3],[1,2,3]])
ar3 = ar1 * ar2
print(ar3)
```

```
[[1 4 9]
 [1 4 9]]
```

```
# Element-wise arithmetic
ar1 = np.array([[1,2,3],[1,2,3]])
ar2 = ar1 * 10 + 7
print(ar2)
```

```
[[17 27 37]
 [17 27 37]]
```

```
# Element-wise arithmetic
ar1 = np.array([[1,2,3],[1,2,3]])
ar2 = ar1 ** 4
print(ar2)
```

```
[[ 1 16 81]
 [ 1 16 81]]
```

```
ar1 = np.random.randint(0,1000, (2,2))
ar2 = np.random.randint(0,1000, (2,2))
ar3 = ar1 >= ar2
print(ar3)
```

```
[[False False]
 [ True False]]
```

Matrix multiplication

```
# Matrix multiplication (m,n)X(n,z) = (n,n)
ar1 = np.array([[1,2,3],[1,2,3]])
ar2 = np.array([[1,1],[2,2],[3,3]])
ar3 = np.matmul(ar1, ar2)
print(ar3)
```

```
[[14 14]
 [14 14]]
```

Solution for systems of linear equations

```
#  2x + y +  z = 7
#  2x - y + 2z = 6
#  x  -2y +  z = 0
arr1 = np.array([[2, 1, 1], [2, -1, 2], [1, -2, 1]])
arr2 = np.array([7, 6, 0])
# Solution of linar equation X, Y an Z
print(np.linalg.solve(arr1, arr2))
```

```
[1. 2. 3.]
```

Hands on - find the solution for the following

1. Problem 1:

$$X + Z + 2W = 6$$
$$Y - 2Z = -3$$
$$X + 2Y - Z = -2$$
$$2X + Y + 3Z - 2W = 0$$

2. Problem 2:

$$-2x_1 - 10x_2 + 4x_3 = 86$$
$$-9x_1 + 9x_2 - 3x_3 + 3x_4 = -108$$
$$-8x_1 - 7x_2 + 2x_3 - 3x_4 = 39$$
$$-2x_1 + 8x_2 + 8x_3 + 9x_4 = -93$$

Array Slicing

- Slicing means taking parts from the array using given indices

  array_name **[start_index : end_index]**

- We may define the step: **[start_index : end_index : step_size]**
  - If we don't pass start its considered 0 **[: end]**
  - If we don't pass end it will be the length of array **[start :]**
  - If we don't pass step its considered 1

Array Slicing

```
array1 = np.array([1, 3, 5, 7, 8, 9, 2, 4, 6])

# slice array1 from index 2 to index 6 (exclusive)
print(array1[2:6])  # [5 7 8 9]

# slice array1 from index 0 to index 8 (exclusive)
print(array1[0:8:2])  # [1 5 8 2]

# slice array1 from index 3 up to the last element
print(array1[3:])  # [7 8 9 2 4 6]

# items from start to end
print(array1[:])   # [1 3 5 7 8 9 2 4 6]
```

Array Slicing

```
numbers = np.array([2, 4, 6, 8, 10, 12])

print(numbers[-1])    # 12

# slice the last 3 elements of the array using the start parameter
print(numbers[-3:])    # [8 10 12]

# slice elements from 2nd-to-last to 4th-to-last element using the start and stop parameters
print(numbers[-5:-2])    # [4 6 8]

# slice every other element of the array from the end
# using the start, stop, and step parameters
print(numbers[-1::-2])   # [12 8 4]
```

Pandas

- Pandas is a Python library released in 2008
- It is built on top of the NumPy library of Python
- It is used for working with data sets; for analyzing, cleaning, exploring, and manipulating data
- Key components provided by Pandas:
  - Series
  - DataFrame

Series

- A Pandas Series is like a column in a table.
- It is a one-dimensional array like structure with homogeneous data
- It has two parts:
  1. Data part (actual data)
  2. Associated index with data (associated array of indices)

```
import pandas as pd
my_series = pd.Series(['Apple', 'Orange', 'Banana'])
print(my_series)
```

```
0     Apple
1    Orange
2    Banana
dtype: object
```

Specify index for a series

```
import pandas as pd
my_series = pd.Series(['Apple', 'Orange', 'Banana'])
print(my_series)
```

```
0     Apple
1    Orange
2    Banana
dtype: object
```

```
my_series = pd.Series(['Apple', 'Orange', 'Banana'], index=['a', 'b','c'])
print(my_series)
```

```
a     Apple
b    Orange
c    Banana
dtype: object
```

Series from Dictionary

- Key of each tuple becomes index in the series

```
calories_per_day = {"day1": 420, "day2": 380, "day3": 390}
my_series = pd.Series(calories_per_day)
print(my_series)
```

```
day1    420
day2    380
day3    390
dtype: int64
```

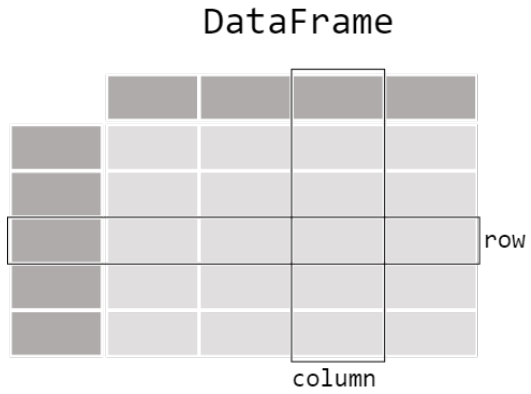
min(), max(), mean(), sum()

- You can apply aggregation functions on the series

```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
my_series = pd.Series(calories)
print('sum=',my_series.sum())
print('min=',my_series.min())
print('max=',my_series.max())
print('mean=',my_series.mean())
print('std=',my_series.std())
```

```
sum= 1190
min= 380
max= 420
mean= 396.6666666666667
std= 20.816659994661325
```

DataFrame



- It is a table of rows and columns
- DataFrame is a multi-dimensional table made up of a collection of Series

Series

	apples
0	3
1	2
2	0
3	1

+

Series

	oranges
0	0
1	3
2	7
3	2

=

DataFrame

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

```
data={'apples':[3,2,0,1],'oranges':[0,3,7,2]}
my_dataframe = pd.DataFrame(data)
print(my_dataframe)
```

```
   apples  oranges
0       3        0
1       2        3
2       0        7
3       1        2
```

```
data={'apples':[3,2,0,1],'oranges':[0,3,7,2]}
my_dataframe = pd.DataFrame(data, index=['SAT', 'SUN', 'MON', 'TUE'])
print(my_dataframe)
```

```
     apples  oranges
SAT       3        0
SUN       2        3
MON       0        7
TUE       1        2
```

Create Dataframe from existing Series

```
equipment_series = pd.Series(['hd', 'keyboard', 'mouse', 'screen'])
quantity_series = pd.Series([10, 54, 36, 12])
stock_df = pd.DataFrame({'Equipment':equipment_series, 'Quantity':quantity_series})
print(stock_df)
```

```
  Equipment  Quantity
0        hd        10
1  keyboard        54
2     mouse        36
3    screen        12
```

```
print(stock_df['Quantity'])
```

```
0    10
1    54
2    36
3    12
Name: Quantity, dtype: int64
```

Create Dataframe from tuples

```
stock = [('hd',10), ('keyboard', 54), ('mouse',36), ('screen',12)]
stock_dataframe = pd.DataFrame(stock)
print(stock_dataframe)
```

```
          0   1
0        hd  10
1  keyboard  54
2     mouse  36
3    screen  12
```

Specify index and columns' names for a dataframe

```
stock = [('hd',10), ('keyboard', 54), ('mouse',36), ('screen',12)]
stock_dataframe = pd.DataFrame(stock, index = ['i1', 'i2', 'i3', 'i4'])
print(stock_dataframe)
```

```
           0   1
i1        hd  10
i2  keyboard  54
i3     mouse  36
i4    screen  12
```

```
stock = [('hd',10), ('keyboard', 54), ('mouse',36), ('screen',12)]
stock_dataframe = pd.DataFrame(stock, index=['i1','i2','i3','i4'], columns=['equipment','quantity'])
print(stock_dataframe)
```

```
   equipment  quantity
i1        hd        10
i2  keyboard        54
i3     mouse        36
i4    screen        12
```

Loc Vs. iLoc

```
print(stock_dataframe)
```

```
    equipment  quantity
i1         hd        10
i2   keyboard        54
i3      mouse        36
i4     screen        12
```

```
print(stock_dataframe.loc['i2'])
```

```
equipment    keyboard
quantity           54
Name: i2, dtype: object
```

```
print(stock_dataframe.iloc[1])
```

```
equipment    keyboard
quantity           54
Name: i2, dtype: object
```

Importing / Exporting Datasets

- Read a comma-separated values (csv) file into DataFrame.
  - Pd.read_csv('FILE_NAME')

- Write DataFrame to a comma-separated values (csv) file.
  - DataFrame.to_csv('FILE_NAME')

Head() and Tail()

- DataFrame.head(*n=5*)
  - Return the first n rows.
  - N is optional, if nothing passed default is 5
- DataFrame.tail(*n=5*)
  - Return the last n rows.
  - N is optional, if nothing passed default is 5

Info() and describe() methods

- DataFrame.**info**()
  - prints a concise summary of a DataFrame.
  - This method prints information about a DataFrame including the index dtype and columns, non-null values and memory usage.
- DataFrame.describe()
  - Generate descriptive statistics.
  - Descriptive statistics include those that summarize the central tendency, dispersion and shape of a dataset’s distribution, excluding NaN values.

Shape and columns properties

- DataFrame.**shape**
  - Return a tuple representing the dimensionality of the DataFrame.
- DataFrame.**columns**
  - Return the column labels of the DataFrame.

Set index

DataFrame.set_index('columns_names')

- Set the DataFrame index using existing columns.
- Set the DataFrame index (row labels) using one or more existing columns or arrays (of the correct length).

Replace method and inplace property

DataFrame.replace(old_Value, new_values, inplace)

- The replace() method searches the entire DataFrame and replaces every case of the specified value.
- **Inplace:** *bool, default False*
  - Whether to modify the DataFrame rather than creating a new one.

Remove duplicate rows

DataFrame.drop_duplicates()

- The drop_duplicates() method removes duplicate rows.

Remove missing values

DataFrame.dropna()

- The dropna() method removes the rows that contains NULL values.

Workshop - The Pandas Framework

- https://www.hifis.net/workshop-materials/python-pandas/