Python for Data Analysis

Short Intro

BBM467 – Data Intensive Applications

Dr. Fuat Akal
akal@hacettepe.edu.tr

Lecture Overview

- Python Libraries to Analyse Data
  - Pandas
  - Numpy
  - Matplotlib

Python Libraries to Analyse Data

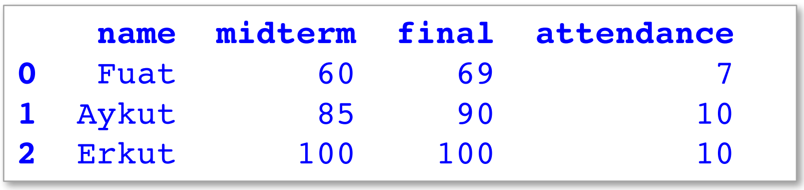
- Pandas
  - Provides data structures and operations for data (e.g. tables and time series) manipulation and analysis.
- Numpy
  - Provides means to work with multidimensional arrays.
- Matplotlib
  - A plotting library used to create high-quality graphs, charts, and figures.

Pandas

- A library that contains high-performance, easy-to-use data structures and data analysis tools.
- Some important aspects of Pandas
  - A fast and efficient DataFrame object for data manipulation with integrated indexing.
  - Tools for reading and writing data in different formats, e.g. csv, Excel, SQL Database.
  - Slicing, indexing, subsetting, merging and joining of huge datasets.
- Typically imported as `import pandas as pd` in Python programs

Create DataFrames using Dictionaries

```
import pandas as pd
data = { 'name': ['Fuat', 'Aykut', 'Erkut'],
         'midterm': [60, 85, 100],
         'final': [69, 90, 100],
         'attendance': [6, 10, 10]
}
df_bbm101 = pd.DataFrame(data)

print(df_bbm101.head()) # Prints top 5 rows
```



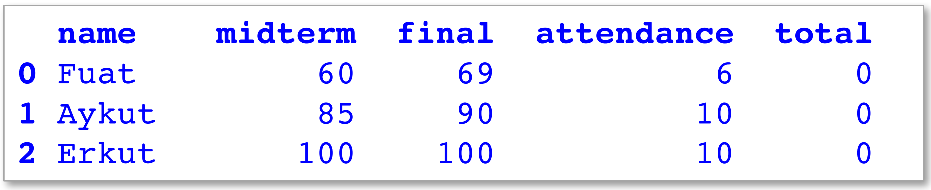
```
     name  midterm  final  attendance
0    Fuat       60     69           7
1   Aykut       85     90          10
2  Erkut      100    100          10
```

Same Thing, in Another Way

```
names = ['Fuat', 'Aykut', 'Erkut']
midterms = [60, 85, 100]
finals = [69, 90, 100]
attendances = [6, 10, 10]

list_labels = ['name', 'midterm', 'final', 'attendance']
list_cols = [names, midterms, finals, attendances]

zipped = list(zip(list_labels, list_cols))

print(zipped)     # [('name', ['Fuat', 'Aykut', 'Erkut']),
                  # ('midterm', [60, 85, 100]),
                  # ('final', [69, 90, 100]),
                  # ('attendance', [6, 10, 10])]

data = dict(zipped)

df_bbm101 = pd.DataFrame(data)
```

Broadcasting

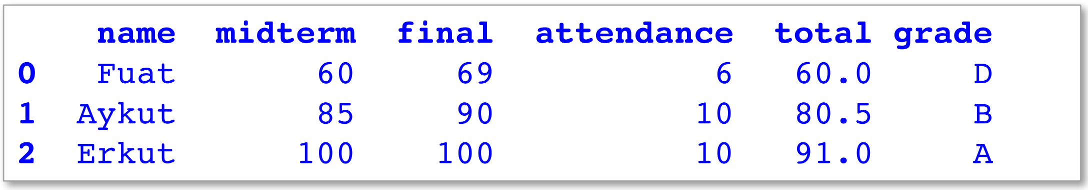
```
df_bbm101['total'] = 0
# Adds new column to df and
# broadcasts 0 to entire column

print(df_bbm101.head())
```



```
    name    midterm  final  attendance  total
0  Fuat         60     69           6      0
1  Aykut        85     90          10      0
2  Erkut       100    100          10      0
```

Compute Columns

```
df_bbm101['total'] = df_bbm101['midterm']*0.3 + \
                     df_bbm101['final']*0.6 + \
                     df_bbm101['attendance']*0.1

df_bbm101.loc[(df_bbm101['total'] >= 60) &
         (df_bbm101['total'] < 70), 'grade'] = 'D’
…            # Code to compute Bs and Cs comes here
df_bbm101.loc[df_bbm101['total'] >= 90, 'grade'] = 'A’

print(df_bbm101.head())
```

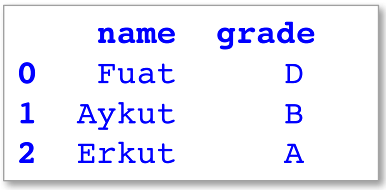


```
    name  midterm  final  attendance  total grade
0   Fuat       60     69           6   60.0     D
1  Aykut       85     90          10   80.5     B
2  Erkut      100    100          10   91.0     A
```

Beware that Fuat would not make it if he missed just one more lecture ;-)

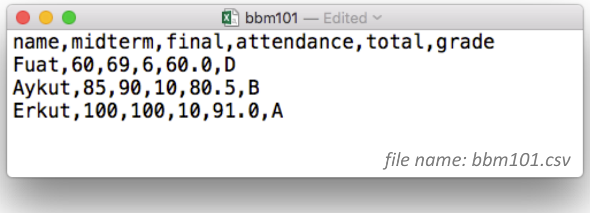
Subsetting/Slicing Data

```
print(df_bbm101[['name', 'grade']])

print(df_bbm101.iloc[:, [0, 5]])

print(df_bbm101.iloc[:, [True, False, False, False,
                                 False, True]])

# They all return the same thing
# name and grade columns of the df
# Same principle can be applied to rows as well
```



```
     name grade
0    Fuat     D
1  Aykut     B
2  Erkut     A
```

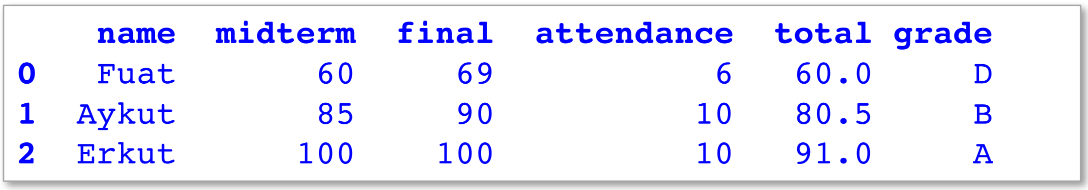
DataFrames from CSV Files



```
name,midterm,final,attendance,total,grade
Fuat,60,69,6,60.0,D
Aykut,85,90,10,80.5,B
Erkut,100,100,10,91.0,A
```

file name: bbm101.csv

```
df_bbm101 = pd.read_csv('bbm101.csv')
print(df_bbm101.head())
```



```
     name  midterm  final  attendance  total grade
0    Fuat       60     69           6   60.0     D
1   Aykut       85     90          10   80.5     B
2   Erkut      100    100          10   91.0     A
```

Indexing DataFrames

```
df_bbm101 = pd.read_csv('bbm101.csv', index_col ='name')
print(df_bbm101.head())
```

```
         midterm  final  attendance  total grade
name
 Fuat         60     69           6   60.0     D
Aykut         85     90          10   80.5     B
Erkut        100    100          10   91.0     A
```

```
print(df_bbm101.loc['Fuat'])
```

```
midterm         60
final           69
attendance       6
total           60
grade            D
Name: Fuat, dtype: object
```

```
print(df_bbm101.
    loc[['Aykut', 'Erkut']])
```

```
         midterm  final  attendance  total grade
name
Aykut         85     90          10   80.5     B
Erkut        100    100          10   91.0     A
```

Numpy

- A library for the Python programming language, adding support for large **multi-dimensional arrays and matrices**,
  - along with a large collection of high-level mathematical functions to operate on these arrays.
- A numpy array is a grid of values, **all of the same type**, and is indexed by a tuple of nonnegative integers.
- The number of dimensions is the **rank** of the array.
- The **shape** of an array is a tuple of integers giving the size of the array along each dimension.
- Typically imported as `import numpy as np` in Python programs

Creating Numpy Arrays

```
import numpy as np

a = np.array([1,2,3])     # Create a rank 1 array
print(type(a))            # <class 'numpy.ndarray'>
print(a.shape)            # (3,)
print(a)                  # [1 2 3]
print(a[0], a[1], a[2])   # 1 2 3


b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # (2, 3)
print(b)                           # [[1 2 3]
                                   #  [4 5 6]]
print(b[0, 0], b[0, 1], b[1, 0])   # 1 2 4
```

Miscellaneous Ways to Create Arrays

```
a = np.zeros((2,2))    # Create an array of all zeros
print(a)               # [[ 0.   0.]
                       #  [ 0.   0.]]

b = np.ones((1,2))     # Create an array of all ones
print(b)               # [[ 1.   1.]]

c = np.full((2,2), 7)  # Create a constant array
print(c)               # [[ 7.   7.]
                       #  [ 7.   7.]]

d = np.eye(2)          # Create a 2x2 identity matrix
print(d)               # [[ 1.   0.]
                       #  [ 0.   1.]]

e = np.random.random((2,2))  # Create an array filled with
                             # random values
print(e)                     # Might print
                             # [[ 0.91940167  0.08143941]
                             #  [ 0.68744134  0.87236687]]
```

Indexing Arrays

- Slicing
- Integer Indexing
- Boolean (or, Mask) Indexing

Slicing

- Similar to slicing Python lists.
- Since arrays may be multidimensional, you must specify a slice for each dimension of the array.
- Slices are views (not copies) of the original data.

Slicing Examples

```
a = np.array([[1, 2, 3, 4],     # Create a rank 2 array
              [5, 6, 7, 8],     # with shape (3, 4)
              [9, 10, 11, 12]])

print(a)                        # [[ 1   2   3   4]
                                #  [ 5   6   7   8]
                                #  [ 9  10  11  12]]

b = a[:2, 1:3]
print(b)                        # [[ 2   3 ]
                                #  [ 6   7 ]

print(a[1, :])                  # [5 6 7 8]

print(a[:, :-2])                # [[ 1   2]
                                #  [ 5   6]
                                #  [ 9  10]]
```

Integer Indexing

- NumPy arrays may be indexed with other arrays.
- Index arrays must be of integer type.
- Each value in the array indicates which value in the array to use in place of the index.
- Returns a copy of the original data.

Integer Indexing Examples

```
a = np.array([1, 2, 3, 4, 5, 6])
print(a)                                # [1 2 3 4 5 6]
print(a[[1, 3, 5]])                     # [2 4 6]


a = np.array([[1, 2], [3, 4], [5, 6]])
print(a)                                # [[ 1  2 ]
                                        #  [ 3  4 ]
                                        #  [ 5  6 ]]

# The returned array will have shape (3,)
print(a[[0, 1, 2], [0, 1, 0]])                 # [1 4 5]
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))   # [1 4 5]

# The same element from the source array can be reused
print(a[[0, 0], [1, 1]])                       # [2 2]
print(np.array([a[0, 1], a[0, 1]]))            # [2 2]
```

Boolean (or, Mask) Indexing

- Boolean array indexing lets you pick out arbitrary elements of an array.
- Frequently used to select the elements of an array that satisfy some condition.
  - Thus, called the mask indexing.

Boolean (or, Mask) Indexing Examples

```
a = np.array([1, 2, 3, 4, 5, 6])

bool_idx = (a > 2)
# Find the elements of a that are bigger than 2;
# this returns a numpy array of Booleans of the same
# shape as a, where each slot of bool_idx tells
# whether that element of a is > 2.

print(bool_idx)             # [False False  True
                            #              True  True  True]

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True
# values of bool_idx
print(a[bool_idx])          # [3 4 5 6]

# We can do all of the above in a single concise statement:
print(a[a > 2])             # [3 4 5 6]
```

Array Math

- Basic mathematical functions operate elementwise on arrays.

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

# Elementwise sum
print(x + y)
print(np.add(x, y))
# [[ 6  8]
#  [10 12]]

# Elementwise product
print(x * y)
print(np.multiply(x, y))
# [[ 5 12]
#  [21 32]]
```

Same principle holds for “np.divide, /” and “np.subtract, -”

Array Math (Cont'd)

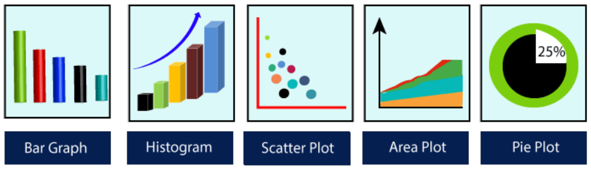
```
x = np.array([[1, 2], [3, 4]])
y = np.array([[5, 6], [7, 8]])

v = np.array([9, 10]
w = np.array([11, 12])
```

```
# Inner product of vectors;
# both produce 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product;
# both produce the rank 1
# array [29 67]
print(x.dot(v))
print(np.dot(x, v))
```

```
# Matrix / matrix product;
# both produce a rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))

# Transpose of x
# [[1 3]
#  [2 4]]
print(x.T)
```

Matplotlib

- Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments.
- Typically imported as `import matplotlib.pyplot as plt` in Python programs.
- Pyplot is a module of Matplotlib which provides simple functions to add plot elements like lines, images, text, etc.
- There are many plot types. Some of are more frequently used.

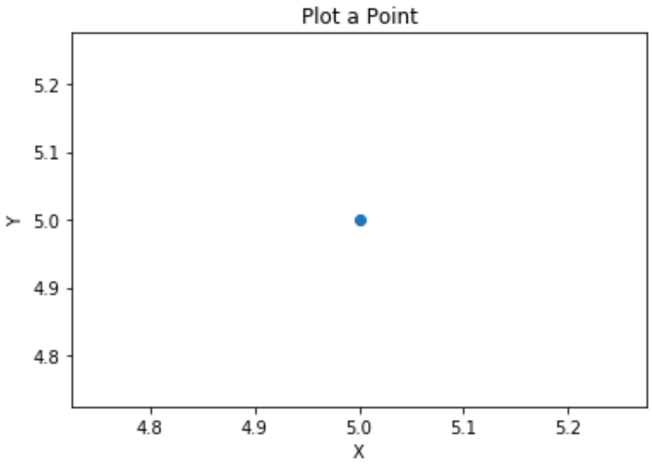


Why Build Visuals?

- For exploratory data analysis
- Communicate data clearly
- Share unbiased representation of data
- A picture is worth a thousand words ☺

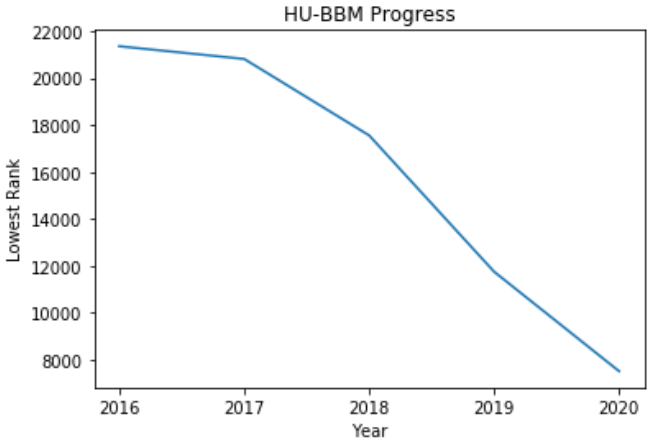
Make a Simple Plot

```
import matplotlib.pyplot as plt

plt.plot(5, 5, 'o')

plt.title("Plot a Point")

plt.xlabel("X")
plt.ylabel("Y")

plt.show()
```



Plot a Simple Line

```
import matplotlib.pyplot as plt

year = ['2016', '2017', '2018', '2019', '2020']
lowest_rank = [21358, 20816, 17555, 11743, 7500]

plt.plot(year, lowest_rank)

plt.title(”HU-BBM Progress")
plt.xlabel('Year')
plt.ylabel('Lowest Rank')

plt.show()
```


HU-BBM Progress
Lowest Rank
Year