# DSC 5101 – Computer Programming in DSAI

## Part II - Working with Data
## HO 05 - Data manipulation using Pandas

Opim Salim Sitompul

Department of Data Science and Artificial Intelligence
Universitas Sumatera Utara

Co-funded by the
Erasmus+ Programme
of the European Union

# Outline I

1 Introduction to Pandas

2 Pandas Objects
- Pandas Series Objects
- Pandas DataFrame Objects
- Pandas Index Objects

3 Data Indexing and Selection

# Introduction to Pandas

- Pandas stands for *’Python Data Analysis’*
- Pandas is built on top of NumPy.
- Provides an efficient implementation of a *DataFrame*
    - *DataFrames* are essentially multidimensional arrays with attached rows and columns labels,
    - often with heterogeneous types and/or missing data.
- Offered convenient data storage interface for labeled data.
- Implements a number of powerful data operation for database frameworks and spreadsheet programs [1].

# Introduction to Pandas

- To use pandas, imports the library:

  import pandas as pd
  pd.__version__ → '1.2.1'

# Pandas Objects

- Pandas objects are enhanced versions of NumPy structures arrays where rows and columns are identified with labels not only simple integer indices.

| NumPy | Pandas |
|---|---|
| simple integer | labels |

- Pandas has three fundamental data structures:
  - Series
  - DataFrame
  - Index

# Pandas Series Objects

- Pandas series is a one-dimensional array of indexed data.

```
import pandas as pd
import numpy
from numpy import random
data = pd.Series(random.random(4))
print(data) → 0 0.703140
              1 0.802721
              2 0.902109
              3 0.824393
              dtype: float64
```

- the Series show a sequence of values and a sequence of indices.
- can be accessed using attributes *values* and *index*.

# Pandas Series Objects

- The *values* are a NumPy array:

  data.values
  array([0.703140, 0.802721, 0.902109, 0.824393])

- The *index* is an array-like object of type pd.Index.

  data.index → RangeIndex(start=0, stop=4, step=1)
  data[1:3] → 1 0.802721
  2 0.902109
  dtype: float64

# Pandas Series Objects

- Series as generalized NumPy array
  - The Pandas Series has an explicitly defined index associated with values

```
data = pd.Series(random.random(4),
                 index=['a', 'b', 'c', 'd'])
data → a 0.678834
        b 0.769742
        c 0.660645
        d 0.429592
        dtype: float64
data['b'] → 0.7697419857496985
```

  - Noncontiguous or nonsequential indices could also be used, for example: index=[2, 5, 3, 7].

# Pandas Series Objects

- Series as specialized dictionary
  - A dictionary is a structure that maps arbitrary key to a set of arbitrary values.
  - A Series is a structure that maps typed keys to a typed values.
- Example:

```
EPL_winner = {'year ': [2018 , 2019 , 2020],
'team ': ['ManchesterCity', 'ManchesterCity', 'Liverpool']}
data = pd.Series(EPL_winner)
data → year [2018, 2019, 2020]
       team [ManchesterCity, ManchesterCity, Liverpool]
        dtype: object
```

# Pandas DataFrame Objects

- DataFrame as a generalized NumPy array
  - DataFrame is an analog two-dimensional array with both flexible indices and flexible column names.

```
winner_dict = {’ManCity1’: 2018, ’ManCity2’: 2019 , ’Liverpool’: 2020}
winner = pd.Series(winner_dict)
win_dict = {’ManCity1’: 32, ’ManCity2’: 32, ’Liverpool’: 32}
win = pd.Series(win_dict)
draw_dict = {’ManCity1’: 4, ’ManCity2’: 2, ’Liverpool’: 3}
draw = pd.Series(draw_dict)
```

# Pandas DataFrame Objects

```
    losses_dict = {'ManCity1': 2, 'ManCity2': 4, 'Liverpool': 3}
    losses = pd.Series(losses_dict)
    EPL_table = pd.DataFrame({'Year': winner, 'Wins': win,
'Draws': draw, 'Losses': losses})
    EPL_table →
```

```
             Year   Wins   Draws   Losses
 ManCity1    2018   32     4       2
 ManCity2    2019   32     2       4
 Liverpool   2020   32     3       3
```

# Pandas DataFrame Objects

- DataFrame as specialized dictionary
  - DataFrame maps a column name to a Series of column data.
  - Asking for champion of the years returns the name of the clubs.

```
print(EPL_table['Year']) →
ManCity1    2018
ManCity2    2019
Liverpool   2020
Name: Year, dtype: int64
print(EPL_table['Wins']) →
ManCity1    32
ManCity2    32
Liverpool   32
Name: Wins, dtype: int64
```

# Pandas DataFrame Objects

- Creating DataFrame Objects
  - From a single Series object

```
clubs = pd.DataFrame(winner, columns=[’Year’])
print(clubs) →
```

```
            Year
ManCity1    2018
ManCity2    2019
Liverpool   2020
```

# Pandas DataFrame Objects

- Creating DataFrame Objects
  - From a list of dict

```
data = [losses_dict for i in range(3)]
print(pd.DataFrame(data)) →
```

```
    ManCity1  ManCity2  Liverpool
0          2         4          3
1          2         4          3
2          2         4          3
```

# Pandas DataFrame Objects

- Creating DataFrame Objects
  - With some data missing:

```
table = [{’Leister’: 2016}, win_dict, draw_dict, losses_dict]
print(pd.DataFrame(table, index= [’Year’, ’Wins’, ’Draws’, ’Losses’]))
```

```
        Leicester  ManCity1  ManCity2  Liverpool
Year       2016.0       NaN       NaN        NaN
Wins          NaN      32.0      32.0       32.0
Draws         NaN       4.0       2.0        3.0
Losses        NaN       2.0       4.0        3.0
```

# Pandas DataFrame Objects

- Creating DataFrame Objects

```
def main():
  data = pd.read_csv("C:/users/Opim Salim Sitompul/football.csv")
  EPL_table = pd.DataFrame(np.array(data), columns=(’year’, ’team’, ’wins’, ’draws’, ’losses’, ’GF’, ’GA’))
  EPL_table[’GD’] = EPL_table[’GF’] - EPL_table[’GA’]
  win = EPL_table.loc[:,"wins":"losses"]
  points = calculate_points(win)
  EPL_table[’Points’] = np.array(points)
  print(EPL_table)
```

# Pandas DataFrame Objects

- A csv file is read from a directory folder and saved in a variable data.
- Using NumPy array, data is converted into Pandas DataFrame, and giving the column names.
- An additional column ’GD’ is added from substraction of two existing columns ’GF’ and ’GA’.
- A function to calculate points is called and then added to the table.

```
def calculate_points(EPL_table):
pt_system = pd.DataFrame([3, 1, 0])
points = pd.DataFrame(np.dot(EPL_table,
    pt_system))
return (points)
```

# Pandas Index Objects

- The *Series* and *DataFrame* objects contain an explicit *index* to reference and modify data. (*see* line 4 of function main)

  EPL_table['GD'] = EPL_table['GF'] - EPL_table['GA']
- Index can be thought as an *immutable array* or an *ordered set*.

# Pandas Index Objects

- Creating DataFrame Objects
  - Index as an *immutable array*

```
import pandas as pd

def main():
  data = pd.read_csv("https://archive.ics.
      uci.edu/ml/machine-learning-databases/
      forest-fires/forestfires.csv")
  print(data.head())
  ind = pd.Index([2, 3, 8, 9, 10, 11])
  print(ind.size, ind.shape, ind.ndim, ind.
      dtype)
  print(ind[::2])
```

# Pandas Index Objects

- Source data is retrieved from the UCI Machine Learning repository for Forest Fires Dataset.
- From 13 attributes avalaible, only 6 attributes are assigned as index.
- the size, shape, dimension and type of index could be obtained.
- Index prints only for 2, 8, 10.
- Any attempts to modify the index, will raise an error:
  ind[2] = 0
  raise TypeError("Index does not support mutable operations")

# Pandas Index Objects

- Creating DataFrame Objects
  - Index as an *ordered set*

```
ind_a = pd.Index([2, 3, 8, 9, 10, 11])
ind_b = pd.Index([0, 3, 8, 9, 10, 12])
print(ind_a & ind_b) →
Int64Index([3, 8, 9, 10], dtype='int64')
print(ind_a | ind_b) →
Int64Index([0, 2, 3, 8, 9, 10, 11, 12], dtype='int64')
print(ind_a ∧ ind_b) →
Int64Index([0, 2, 11, 12], dtype='int64')
```

# Data Indexing and Selection

- Methods and tools to access, set, and modify values in NumPy array could also be performed on Pandas Series and DataFrame.
- Example:

```
arr = np.random.normal(0, 1, (3, 3))
arr →
array([[-0.62605847, -0.91465039, 1.60655292],
       [-0.89083616, -0.06934873, -1.45449544],
       [-0.08016883, 0.67681558, 0.57810467]])
arr[2, 1] → 0.6768155816429358 #indexing
arr[:,0:1] → #slicing
array([[-0.62605847, -0.91465039],
       [-0.89083616, -0.06934873],
       [-0.08016883, 0.67681558]])
```

# Data Indexing and Selection

- Data Selection in Series

```
    data = pd.Series(([random.random()] for i in range(5)),
index=[’a’, ’b’, ’c’, ’d’, ’e’])
    data →
    a    [0.29643029563914247]
    b    [0.004881720836473313]
    c    [0.3093878397731965]
    d    [0.5531367830523516]
    e    [0.567471553876928]
    dtype object
    data[’b’] →
    [0.004881720836473313]
```

# Data Indexing and Selection

- Dictionary-like Python:

  'a' in data →
  True
  data.keys() →
  Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
  data['f'] = [1.1378]
  list(data.items()) →
  [('a', [0.29643029563914247]), ('b', [0.004881720836473313]), ('c', [0.30938783973731965]), ('d', [0.5531367830523516]), ('e', [0.567471553876928]), ('f', [1.1378])]

# Data Indexing and Selection

- Series as one-dimensional array:

```
data[’a’:’c’] →
a    [0.29643029563914247]
b    [0.004881720836473313]
c    [0.30938783973731965]
data[’a’, ’e’] →
a    [0.29643029563914247]
e    [0.567471553876928] dtype: object
```

# Data Indexing and Selection

- Indexer: loc, and iloc

  data.loc['a':'c'] →

  ```
  a    [0.29643029563914247]
  b    [0.0048817208364733l3]
  ```

  data.iloc[1] →

  [0.0048817208364733l3]

# Data Indexing and Selection

- data = pd.read_csv("https://archive.ics.uci.edu/ml/machine-learning-databases/forest-fires/forestfires.csv")

  data.iloc[:2,8:11] →

```
     temp   RH   wind
0     8.2   51    6.7
1    18.0   33    0.9
```

  df.iloc[:2,0] →

```
0   7
1   7
```

# References I

[1] VanderPlas J., Python Data Science Handbook: Essential Tools for Working with Data, O'Reilly, USA, 2016.

[2] Hunt, J., A Beginners Guide to Python 3 Programming, Springer Nature Switzerland AG, Switzerland, 2019.

[3] Hunt, J., Advance Guide to Python 3 Programming, Springer Nature Switzerland AG, Switzerland, 2019.