Department of Computer Engineering

University of Peradeniya

CO544 Machine Learning and Data Mining

Lab 02

06th April 2020

Objective:

To provide students hands on experience on Python modules, NumPy and Pandas.

1. Introduction

NumPy, short for Numerical Python, is the fundamental package required for high-performance scientific computing and data analysis. It provides a high-performance multidimensional array object, and tools for working with these arrays. Most of the frameworks and libraries such as Scikit-Learn, Tensorflow which require numerical calculation use NumPy.

Pandas contains high-level data structures and manipulation tools designed to make data analysis fast and easy in Python. It offers data structures and operations for manipulating numerical tables and time series. This is built on top of NumPy and makes it easy to use in NumPy-centric applications.

2. NumPy

2.1 Creation

```
import numpy as np  # import numpy module as np

a=np.array([1,2,3]) # Creating 1D array
a.dtype  # return the data type of the array
matrix = np.array ([np.arange (3), [i for i in range(1 ,4)],[6 ,7 ,8]])
```

2.2 Initialization

```
np.zeros((5,2,2),dtype=float) # array of all zero of float data type
np.ones(4,5) # array full of one's
np.empty([3,4]) #array which initial content is random
np.arange (2 ,10 ,2) # array with evenly spaced values
np.arange (2 ,10 ,1).reshape(4,2) #rearranging the size of the array
np.full ([2 ,3] , 4) # creates an array with constant values
np.eye(3) # creates an identity matrix
np.linspace (2 ,3,5) # creates an evenly spaced array within specified
   interval
```

2.3 Copying, Sorting, Slicing

```
np.copy(matrix) #returns the copy of the object
matrix.copy() #deep copy
matrix.view() #shallow copy
matrix.sort() # sorts in ascending order
matrix.sort(axis=1) #sort along the specified axis
matrix [0: ,:1]) # 2D array slicing
matrix [:2, 0:2])
matrix [:1, :])
```

2.3.1 Try out

```
matrix [1,0]
matrix [0] = 42
matrix [1:3]
matrix []
matrix [1:]
matrix [1:100]
matrix [:]
matrix [1: ,:2]
matrix [:2, 1:]
matrix.ravel ()
matrix [: ,1]. copy ()
matrix [1]. tolist ()
matrix.reshape(-1)
```

```
>>> a[0, 3:5]
array([3, 4])

>>> a[4:, 4:]
array([[44, 55],
       [54, 55]])

>>> a[:, 2]
a([2, 12, 22, 32, 42, 52])
```

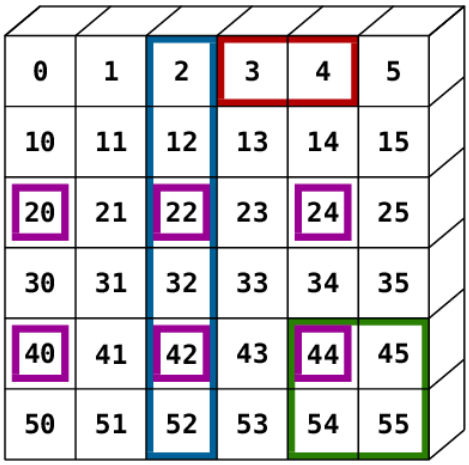

Figure 1

Source: scipy-lectures.org

2.4 Operations and Functions

Arithmetic operators and universal functions (sin, cos, exp) operates element wise for arrays and produce an array with results.

2.4.1 Try out

```
np.sqrt(matrix)
np.exp(matrix)
np.min(matrix)
np.max(matrix, axis=1)
np.min(np.maximum(np.random.randn(4),np.random.randn(4)))
np.mean(matrix)
np.mean(matrix, axis=0)
np.sum(matrix)
np.invert(matrix)
np.random.randn(5)
np.trace(matrix)
```

Hope by now, you have the basic understanding about how to deal with NumPy. Try to solve this problem. Let's assume you are to implement basic version of random walk. Walk would start at any point (e.g. 0 or 10) step of 0 or 1. Current position should be deducted by 1 for 0 value occurrence. Try to implement single random walk with 500 steps.

3. Python Classes

3.1 Class Definition Syntax

```
class RandomWalk :
<statement-1>
.
.
.
<statement-N>
```

3.2 Class Objects

```
class RandomWalk :
def __init__[self, position]:
self.position = position
def walk(self):
return walked path
random walker = RandomWalk (200)
```

4. Pandas

4.1 Importing Pandas

```
import pandas as pd # import pandas module as pd
```

4.2 Series and Data Frames

Series and Data Frames are the primary objects which provided by Pandas. Series is a one-dimensional labeled array capable of holding any data type with indexing capabilities. When it is required more than one Series of data that is aligned by a common index pandas DataFrame can be employed. A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

4.2.1 Creating Series and Data Frames

```
# create a series with a list
s = pd.Series([1,4,-2,'home'],index=['a','b','c','d'])
```

ToDo 1: What is the data type of s? Can it be changed?

```
# create a data frame with a dictionary
data={'population':[1.5,1.2,2.0,1.4,0.8],'state':['Nevada','Florida','O
 hio','Texas','Florida'],'year':[2003,2000,2004,1990,1994]}
```

```
df=pd.DataFrame(data,index=['one','two','three','four','five'],columns=
['year','state','population','debt'])
```

4.2.2 Accessing and modifying

```
s[1:3]
s[0]
s['d']
s.values[2:]
df[['population','state']]
df.population
df.iloc[1:]
df.iloc[2:4:,2:5]
df.loc['one']
df.debt=34.67
df.debt=[df.iloc[:,2][i]*5 for i in range(0,df.shape[0])]
df.head()
df.tail(2)
df.sample(n=3)
df['newColomn']=pd.Series(np.random.randn(df.shape[0]),index=df.index
df.drop_duplicates('state')
df.state
```

4.2.3 Loading data from CSV file

```
df=pd.read_csv('sampleDataSet.csv') # without setting names
df=pd.read_csv('sampleDataSet.csv’,names=['a','b','c','d','e','f','g','
h','i']) # setting names
```

ToDo 2: Comment on the shape of the data frame with and without setting names.

4.2.4 Dealing with missing values.

```
df.isnull().g
df.isnull().sum(0)
df=df[df.isnull().a != True]
df.dropna(axis=0).isnull().sum()
df.dropna(axis=1)
df.dropna(axis=1, how='all')
df.dropna(axis=1, thresh=1)
df.drop('i',axis=1)
df.fillna(899)
df.fillna(method='ffill')
df.replace(6.3,600)
df.replace('.',np.nan)
df[np.random.rand(df.shape[0]>0.5)]=1.5
```

4.2.5 Applying functions

Functions can be written using ‘lambda’ expression or using ordinary function definition.

```
f=lambda df: df.max()-df.min()
def f(x): return x.max()-x.min()

df.iloc[: ,3:5].apply(f) # applying function element wise
```

4.2.6 Group Operations

```
grouped=df[['a','b','e']].groupby(df['i']) #group according to column ‘i’
grouped.mean()
grouped=df[['a','b','e']].groupby([df['i'],df['c']]).mean()
grouped.unstack()
```

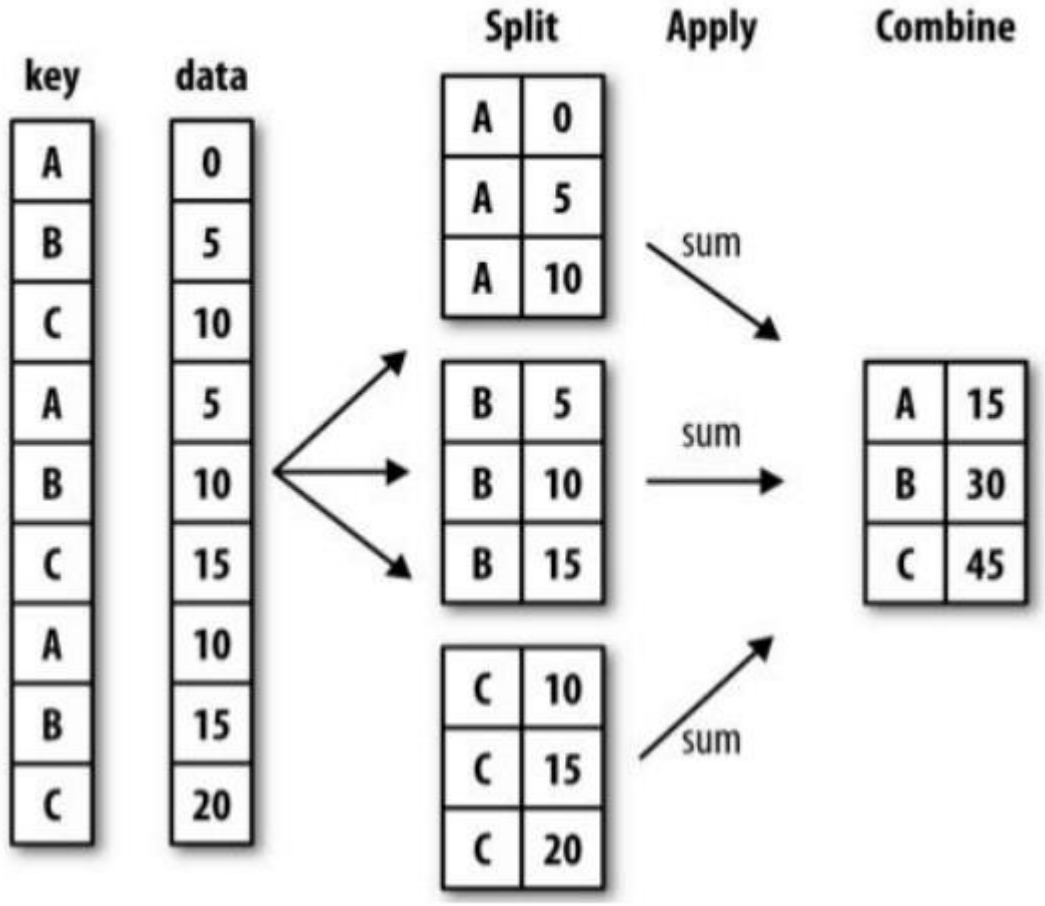

Figure2: Group Aggregation

4.2.7 Data Summarizing

```
df['a'].nunique() # number of distinct values in a column
df['a'].value_counts() # count the number of rows for each unique value
df.describe() # descriptive statistics for each column
df.mean()
df.sort_index().head()
```

4.2.8 Data Visualization

```
df.plot(kind='hist')
df.plot(kind='bar')
df.boxplot()
```

4.3 Try Out

Data wrangling is the process of transforming and mapping data from raw data into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics. Once you proceed with below steps you would able to understand how Pandas can be used for data wrangling.

1. Load the Lab02Ecercise01.csv file and specify the column names as : Chanel1, Chanel2, Chanel3, Chanel4 and Chanel5
2. If there's any missing values fill them with the mean value of corresponding column.
3. To see the correlation between one column to all other columns, use following code segment and comment on diagonal plot.

```
from pandas.plotting import scatter_matrix
scatter_matrix (data , alpha =0.2 , figsize =(6, 6),diagonal='kde')
```

4. Add a new column named as "class" on lab02Exercise02 dataset. Values of this column either 1 or 0 which should be derived based on the following condition:

   if ((Colomn1 + Colomn5)/2) < ((Colomn2 + Colomn3 + Colomn4)/3) then
   value ← 1
   else
   value ← 0
   end if

5. Lab Exercise

You have to practice all the commands and exercises in the lab and implement random walker in the section 2.4.1 using NumPy package and Data wrangling using Pandas in section 4.3 for the submission.

6. Submission

Submit two text files for two sections (i:e NumPy and Pandas). Your files should contain all your commands you have tried out in the lab with the answers for TODO sections (as comments in the text) and Try Outs .Rename it as 15xxxlab02np.txt and 15xxxlab02pd.txt where xxx is your registration number.

7. Deadline

The deadline: April 16, 2020, by 4 p.m.

8. Note

Make sure that now you have the basic understand what we did during the lab. This lab is really important for successive labs as well. If you do not understand any concepts, make sure you get some help from instructors.