Fourth Industrial Summer School

Day 1

Introduction to Data Preparation

Session Objectives

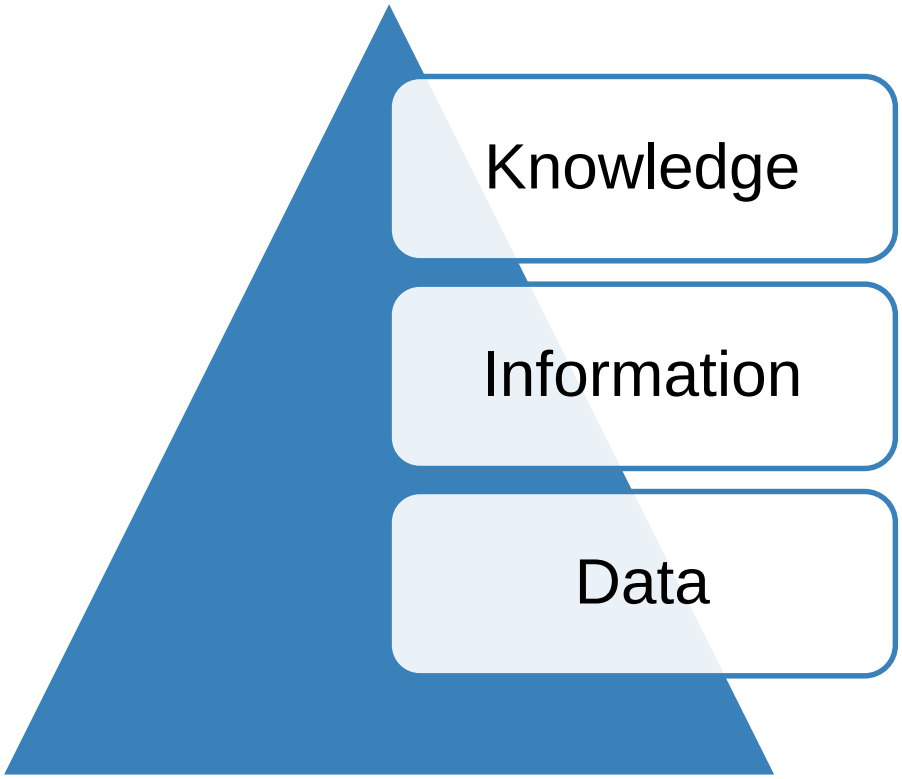
- ✓ Knowledge discovery process
- ✓ Why Prepare Data?
- ✓ Types of Data

- ✓ Python for Data Analysis
  - Python Libraries
  - NumPy arrays
  - Pandas Series and DataFrame.

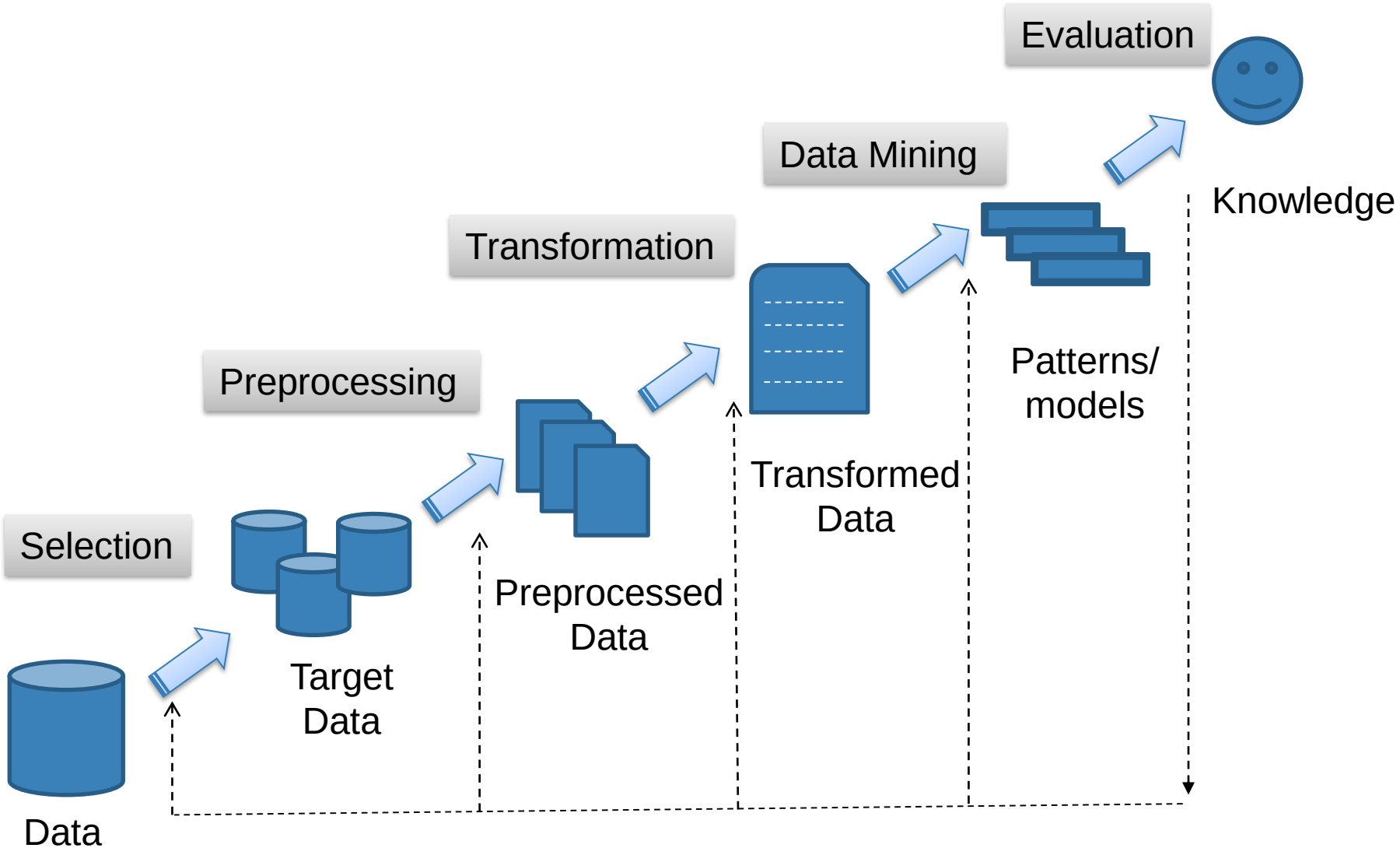
Introduction

Data Preparation – Crucial Step of Data Science Process

Introduction



Knowledge discovery process



Why Prepare Data?

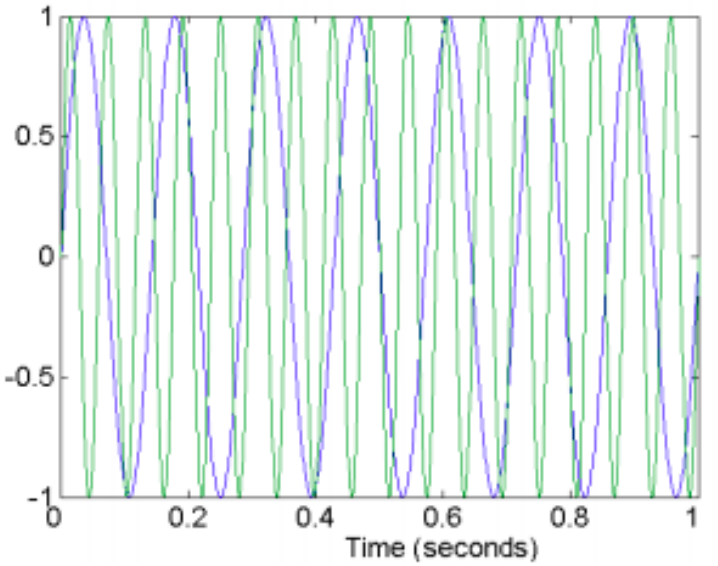
- The data that you access from various sources doesn't come in an easily packaged form, ready for analysis.
- You may also need to transform it to make all the data sources cohesive and amenable to analysis.
- Transformation may require changing data types, the order in which data appears, etc.
- Data might contain outliers, missing values, or errors.

Need for Data Preparation

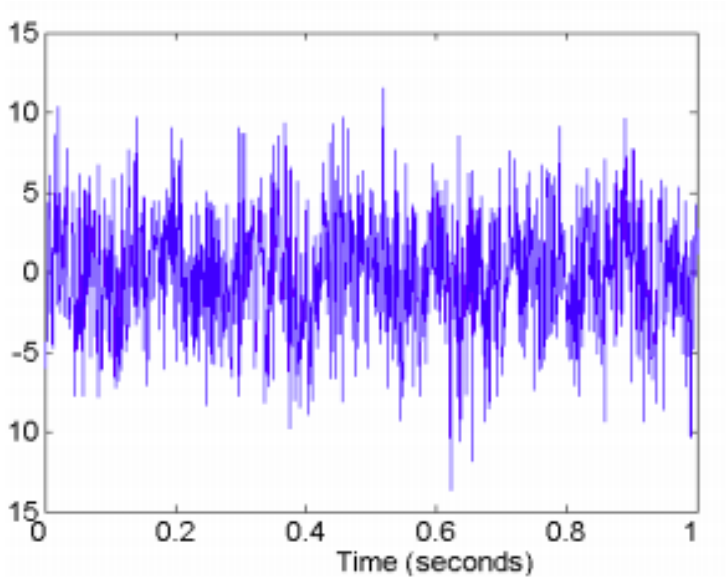
- Correct data preparation prepares both the miner and the data.
- The quality of the built models depend on
  - the content of the data
  - the ability of the modeler.
- Thus, preparing data also prepares the miner
  - using prepared data the miner produces better models

Noise

- Noise refers to modification of original values
  - Examples: distortion of a person's voice when talking on a poor phone and "snow" on television



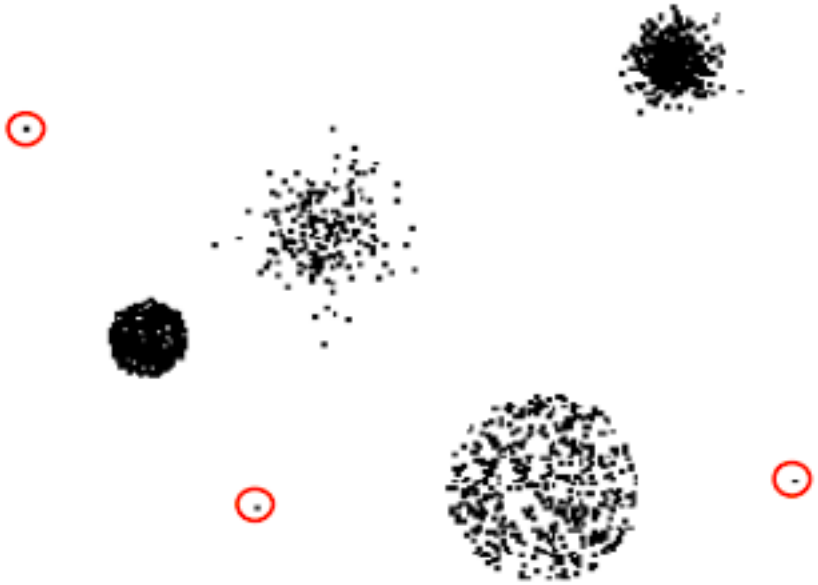
Two Sine Waves



Two Sine Waves + Noise

Outliers

- Outliers are data objects with characteristics that are considerably different than most of the other data objects in the data set



Missing Values

- Reasons for missing values
  - Information is not collected (e.g., people decline to give their age and weight)
- Attributes may not be applicable to all cases
  - (e.g., annual income is not applicable to children)
- Handling missing values
  - Eliminate Data Objects
  - Estimate Missing Values
  - Ignore the Missing Value During Analysis
  - Replace with all possible values (weighted by their probabilities)

Duplicate Data

- Data set may include data objects that are duplicates, or almost duplicates of one another
  - Major issue when merging data from heterogeneous sources

- Examples:
  - Same person with multiple email addresses

Topics in this module

- Types of Data
- Python Libraries for Data Analysis
- Data Loading
- Descriptive Analysis
- Data Preprocessing
- Data Cleaning
- Data Manipulation
- Data Transformation
- Data Aggregation and Grouping

Types of Data

What is Data?

- Collection of data objects and their attributes

Attributes

Objects

City	Hi	Lo	Conditions
Acapulco	99	77	pc
Bangkok	92	78	pc
Mexico	77	57	sh
Montreal	72	56	pc
Paris	77	58	c
Rome	88	68	cl
Toronto	78	61	c

- An attribute is a property or characteristic of an object –
  - Examples: ID , name, etc. –
  - Attribute is also known as variable, field, characteristic, or feature

- A collection of attributes describe an object –
  - Object is also known as record, data point, case, sample, entity, or instance

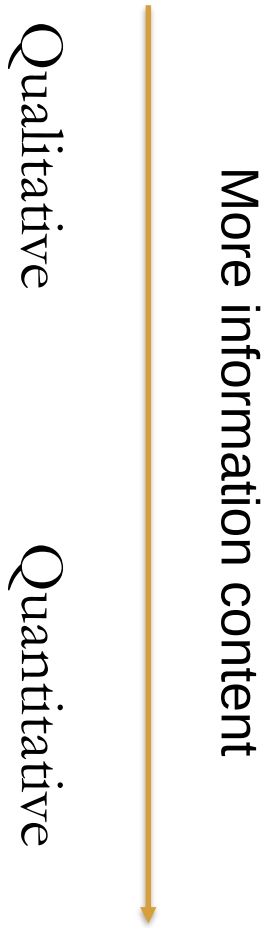
Attribute Values

- Attribute values are numbers or symbols assigned to an attribute

- Distinction between attributes and attribute values
  - Same attribute can be mapped to different attribute values
    - Example: height can be measured in feet or meters
  - Different attributes can be mapped to the same set of values
    - Example: Attribute values for ID and age are integers
    - But properties of attribute values can be different – ID has no limit but age has a maximum and minimum value

Types of Measurements: Examples

- Nominal:
  - ID numbers, Names of people, eye color, zip codes
- Ordinal:
  - rankings (e.g., service on a scale from 1-10), grades, height in {tall, medium, short}
- Interval:
  - calendar dates, temperatures in Celsius or Fahrenheit, GRE and IQ scores
- Ratio:
  - length, time, counts



Properties of Attribute Values

- The type of an attribute depends on which of the following properties it possesses:
  - Distinctness: = ≠
  - Order: < >
  - Addition: + -
  - Multiplication: * /

- Nominal attribute: distinctness
- Ordinal attribute: distinctness & order
- Interval attribute: distinctness, order & addition
- Ratio attribute: all 4 properties

Discrete and Continuous Attributes

- Discrete Attribute –
  - Has only a finite or countably infinite set of values
  - Examples: zip codes, counts, etc.
  - Often represented as integer variables.
  - Note: binary (categorial) attributes are a special case of discrete attributes

- Continuous Attribute
  - Has real numbers as attribute values
  - Examples: temperature, height, or weight
  - Practically, real values can only be measured and represented using a finite number of digits
  - Continuous attributes are typically represented as floating-point variables

Data Conversion

- Some tools can deal with nominal values but other need fields to be numeric

- Convert ordinal fields to numeric to be able to use “>” and “<” comparisons on such fields.
  - A+ → 4.0
  - A → 3.7
  - B+ → 3.3
  - B → 3.0

Exercise

- The following temperatures (high, low) and weather conditions for some selected world cities.

City	Hi	Lo	Conditions
Acapulco	99	77	pc
Bangkok	92	78	pc
Mexico	77	57	sh
Montreal	72	56	pc
Paris	77	58	c
Rome	88	68	cl
Toronto	78	61	c

(a) How many data points are in this data set?
(b) How many variables are in this data set?
(c) How many observations are in this data set?
(d) Which variables are quantitative?
(e) Which attributes are ordinal?

Exercise..

- Determine whether the following statement refers to categorical data, qualitative data, quantitative data, or some combination of these:

  - The instructor gave a test on which the maximum possible score was 100. The actual scores students received were 92, 87, 86, 85, 72, 70, 70, and 61.

    Answer: Quantitative.

  - The instructor gave an exam on which one student received an A, three students received B's, three students received C's, and one student received a D.

    Answer: Qualitative and categorical.

  - Five students had blue backpacks, ten students had red backpacks, and three students had green backpacks.

    Answer: Qualitative and categorical.

Python for Data Analysis

Python Libraries for Data Science

- Many popular Python toolboxes/libraries:
  - Pandas
  - NumPy
  - SciPy
  - SciKit-Learn

- Visualization libraries
  - matplotlib
  - Seaborn

- and many more …

Pandas

- Pandas is a popular Python library for data analysis.
  - It is not directly related to Machine Learning.

- To prepare data before training, Pandas is used for data extraction and preparation. It provides
  - high-level data structures and wide variety tools for data analysis.
  - tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.
  - methods for groping, combining and filtering data.

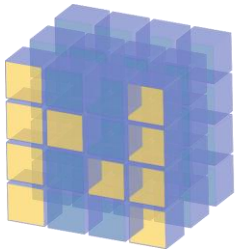
Link: http://pandas.pydata.org/

NumPy

- NumPy (numerical python) is a very popular python library for large multi-dimensional array and matrix processing
  - a large collection of high-level mathematical functions.

- It is very useful for fundamental scientific computations in Machine Learning.

- Many other python libraries are built on NumPy



NumPy

Link: http://www.numpy.org/

SciPy

- SciPy is a python library used for scientific computing and technical computing.
- SciPy contains modules for optimization, linear algebra, integration, signal and image processing and other tasks common in science and engineering.
- SciPy library is one of the core packages that make up the SciPy stack.
- built on NumPy

Link: https://www.scipy.org/scipylib/

Scikit-learn

- Skikit-learn provides machine learning algorithms: classification, regression, clustering, model validation etc.
- It is built on top of two libraries, NumPy and SciPy.
- Scikit-learn supports most of the supervised and unsupervised learning algorithms.

Link: http://scikit-learn.org/

matplotlib

- Matpoltlib is a very popular Python library for data visualization.
- Like Pandas, it is not directly related to Machine Learning.

- It particularly used to visualize the patterns in the data.
- It is a 2D plotting library used for creating 2D graphs and plots.
  - a set of functionalities similar to those of MATLAB
  - line plots, scatter plots, barcharts, histograms, pie charts etc.

Link: https://matplotlib.org/

Seaborn

- Seaborn is a Python data visualization library based on matplotlib.
- It provides a high-level interface for drawing attractive and informative statistical graphics.

- It introduces additional plot types.
- It also makes your traditional Matplotlib plots look a bit prettier.

- Similar (in style) to the popular ggplot2 library in R

Link: https://seaborn.pydata.org/

To remember ..

- Pandas
  - Data manipulation and analysis
- NumPy and Scipy
  - Fundamental scientific computing
- Scikit-learn
  - Machine learning and data Mining
- Matplotlib and Seaborn
  - Plotting and visualization

Importing Python libraries

- To get started, import the required library into your namespace:

```
#Import Python Libraries

import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

- To check the version of a particular library

```
import pandas as pd
print(pd.__version__)
```

```
0.24.2
```

NumPy Arrays

Why do we need NumPy?

- In Numpy, number of dimensions of the array is called rank of the array.

1 D ARRAY:

C	O	D	I	N	G	E	E	K
0	1	2	3	4	5	6	7	8

single row of elements

2 D ARRAY:

	col 0	col 1	col 2
i \ j	0	1	2
row 0 → 0	A	A	A
row 1 → 1	B	B	B
row 2 → 2	C	C	C

column

array elements

rows

Why do we need NumPy?

- Why do we need NumPy when python lists are already there?

```
a = [1,3,5,7,9]
print(a)

b = [[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
print(b)
```

```
[1, 3, 5, 7, 9]
[[1, 3, 5, 7, 9], [2, 4, 6, 8, 10]]
```

- can't use directly with arithmetical operators (+, -, *, /, …)
- can't perform operations on all the elements of two list directly.
  - For example, it is no possible to multiply two lists directly

- This is where the role of NumPy comes into play.

Creation of Arrays

```
# Creation of Arrays
import numpy as np

# Creating a rank 1 Array
arr = np.array([1, 2, 3])
print("Array with Rank 1: \n",arr)
```

```
Array with Rank 1:
 [1 2 3]
```

```
# Creating a rank 2 Array
arr = np.array([[1, 2, 3],
                [4, 5, 6]])
print("Array with Rank 2: \n", arr)
```

```
Array with Rank 2:
 [[1 2 3]
 [4 5 6]]
```

Math operations on arrays

- In Numpy arrays, basic mathematical operations are performed element-wise on the array.

- Some functions can be applied on the whole array e.g. sum, min, max, etc.
  - These functions can also be applied row-wise or column-wise by setting an axis parameter.

- Some others can be applied on each element in the array e.g. sqrt

Math operations on arrays

```
# Defining Array 1
a = np.array([[1, 2],
              [3, 4]])

# Defining Array 2
b = np.array([[4, 3],
              [2, 1]])
```

```
print ("\nSum of all array elements: ", a.sum())
```

```
print ("\nArray sum:\n", a + b)
```

Data Types in Numpy

- Every ndarray has an associated data type (dtype) object.
  - (dtype) object provides information about the layout of the array
- In Numpy, datatypes of Arrays need not to be defined.
  - Numpy tries to guess the datatype for Arrays

```
# Integer datatype
x = np.array([1, 2])
print(x.dtype)

# Float datatype
x = np.array([1.0, 2.0])
print(x.dtype)

# Forced Datatype
x = np.array([1.0, 2.0], dtype = np.int64)
print(x.dtype)
```

```
int64
float64
int64
```

Diagonal and Zero matrix

```
arr= np.diag([1,2,3])
arr
```

```
array([[1, 0, 0],
       [0, 2, 0],
       [0, 0, 3]])
```

```
arr= np.zeros([3,3])
arr
```

```
array([[0., 0., 0.],
       [0., 0., 0.],
       [0., 0., 0.]])
```

Create a random array

- Create an array of the given shape and populate it with random samples from a uniform distribution over [0, 1).

```
r = np.random.rand(3,3)
r
```

```
array([[0.11580128, 0.53542029, 0.59711351],
       [0.02738275, 0.99413621, 0.29678759],
       [0.44860753, 0.41043688, 0.04572588]])
```

Array Access

- To access the first row of two-dim array

```
#Create an array
narr = np.array([[4, 7], [2, 6], [9,0]])

#Access the first row
narr[0]
```

The first row can be accessed also using this notation: **narr[0,:]**

- To access the first column

```
# To access the first column
narr[:, 0]
```

```
array([4, 2, 9])
```

Array Access

```
[44]  # You can also assign to
      # a specific value to entire rwo or column

      narr[1,:] = 10
      narr
```

```
array([[ 4,  7],
       [10, 10],
       [ 9,  0]])
```

Convert the input to an array

- numpy.as array is used to convert a given input to an array

```
a = [[1, 2],[3,4]]
print(a)

b=np.asarray(a)
print(b)
```

```
[[1, 2], [3, 4]]
[[1 2]
 [3 4]]
```

Using np.arange

- create a sequence of numbers then using **np.arange**

```
b = np.arange(start = 20,stop = 30, step = 1)
b
```

```
array([20, 21, 22, 23, 24, 25, 26, 27, 28, 29])
```

In np.arange the end point is always excluded.

- **np.arange** provides an option of step which defines the difference between 2 consecutive numbers.
- If step is not provided then it takes the value 1 by default.

Using np.arange and reshape

- 1D Array

```
import numpy as np

array = np.arange(8)
print("Original array : \n", array)
```

```
Original array :
 [0 1 2 3 4 5 6 7]
```

- 2D Array

```
import numpy as np

# shape array with 2 rows and 4 columns
array = np.arange(8).reshape(2, 4)
print(array)
```

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

Using np.arange and reshape

- 3D Array

```
import numpy as np

# Constructs 3D array
array = np.arange(8).reshape(2, 2, 2)
print(array)
```

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

 [[4 5]
  [6 7]]]
```

Hands-on exercises

Pandas Data Structures

NumPy vs. Pandas

- Pandas is built on top of NumPy.
  - So Pandas is not an alternative to Numpy.

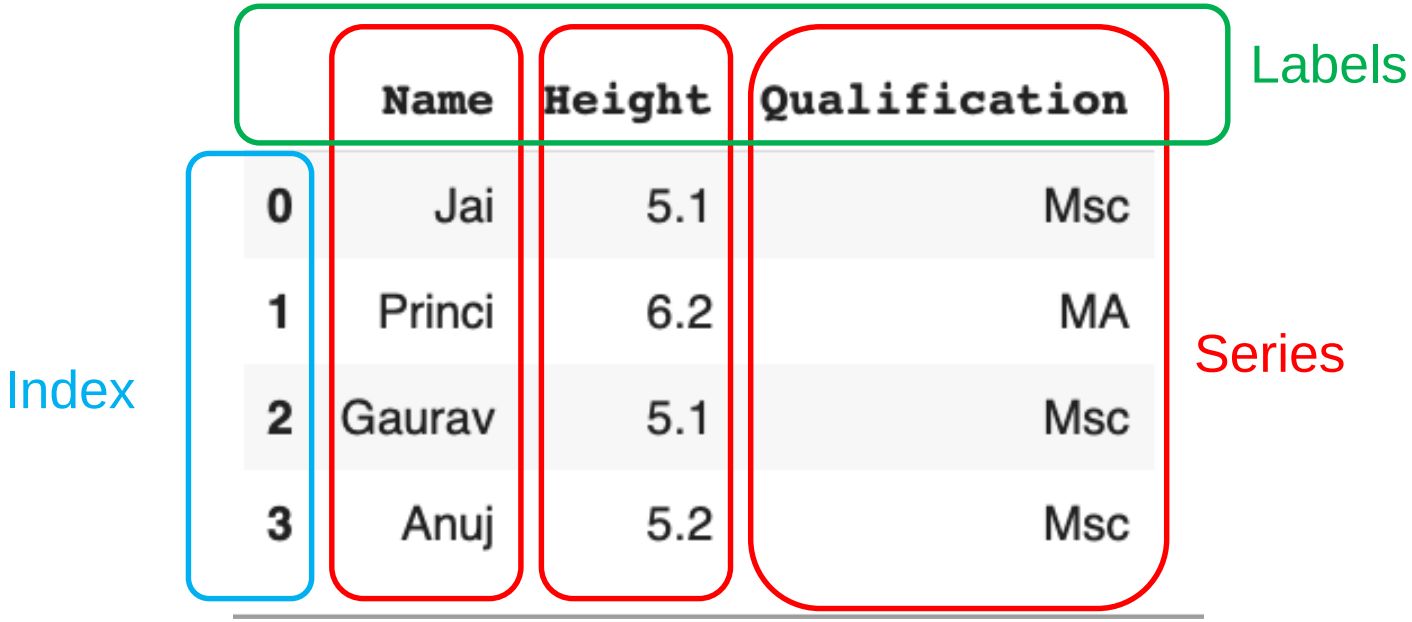
- Instead pandas offers additional method or provides more streamlined way of working with numerical and tabular data in Python.

- Two data structures:
  - Series
  - DataFrame

Series

- Series is a one-dimensional labeled array
  - any data type (integers, strings, float, Python objects, etc.).
  - The axis labels are collectively referred to as the index.
  - It supports both integer and label-based indexing

- Pandas Series is nothing but a column in an excel sheet.



	Name	Height	Qualification
0	Jai	5.1	Msc
1	Princi	6.2	MA
2	Gaurav	5.1	Msc
3	Anuj	5.2	Msc

Creating a Pandas Series

- Create an Empty Series

```
#Declaring a Series

import pandas as pd
s = pd.Series()
print (s)
```

```
Series([], dtype: float64)
```

Creating a Pandas Series

- Creating a series from array:
  - Need to import a numpy module and have to use array() function.

```
# import pandas and numpy
import pandas as pd
import numpy as np

# simple array
data = np.array(['a','b','c','d'])

ser = pd.Series(data)
print(ser)
```

```
0    a
1    b
2    c
3    d
dtype: object
```

Creating a Pandas Series

- You can pass the index to the series

```
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print (s)
```

```
100    a
101    b
102    c
103    d
dtype: object
```

- Note: It is also possible to create a series from a list

Create a Series from dict

- A dict can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index.

- If index is passed, the values in data corresponding to the labels in the index will be pulled out.

```
#simple dict
data = {'a' : 0., 'b' : 1., 'c' : 2.}

#create a series
s = pd.Series(data)
print (s)
```

```
a    0.0
b    1.0
c    2.0
dtype: float64
```

Accessing element of Series

- There are two ways through which we can access element of series, they are:
  - Accessing Element from Series with Position
  - Accessing Element Using Label (index)

```
#retrieve the first element
print s[0]
```

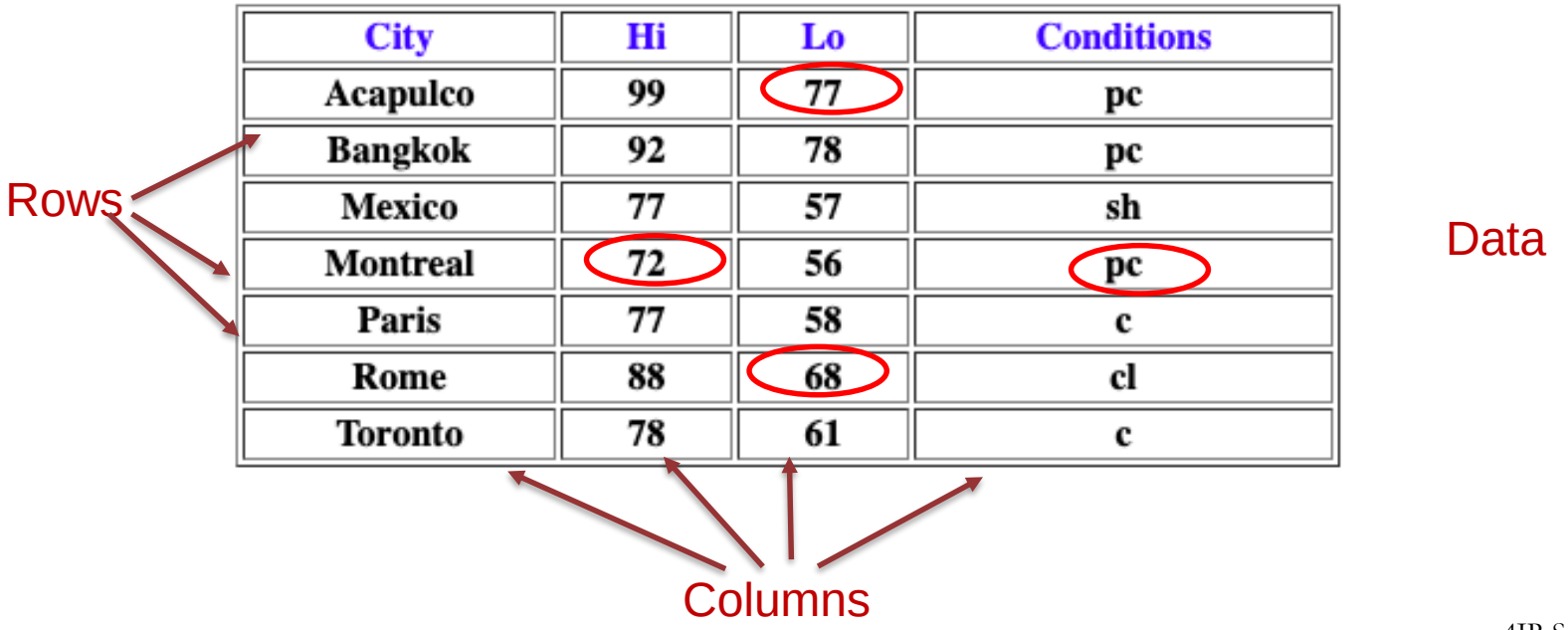
```
# retrieve first three elements
print s[:3]
```

```
# retrieve last three elements
print s[-3:]
```

```
print s[102]
```

Pandas DataFrame

- **Pandas DataFrame** is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns).
- Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.



City	Hi	Lo	Conditions
Acapulco	99	77	pc
Bangkok	92	78	pc
Mexico	77	57	sh
Montreal	72	56	pc
Paris	77	58	c
Rome	88	68	cl
Toronto	78	61	c

Creating a Pandas DataFrame

- In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file.

- Pandas DataFrame can be created in multiple ways.

Creating an empty dataframe

```
# Calling DataFrame constructor
df = pd.DataFrame()

print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

Creating a dataframe using List

- DataFrame can be created using a single list.

```
# list of strings
lst = ['blue','green','yellow','red','white']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
df
```

	0
0	blue
1	green
2	yellow
3	red
4	white

Creating a dataframe using List

- We can create pandas dataframe from lists using dictionary using pandas.DataFrame.
  - With this method in Pandas we can transform a dictionary of list to a dataframe.

```
# dictionary of lists
dict = {'name':["aparna", "pankaj", "sudhir", "Geeku"],
        'degree': ["MBA", "BCA", "M.Tech", "MBA"],
        'score':[90, 40, 80, 98]}

df = pd.DataFrame(dict)
df
```

	name	degree	score
0	aparna	MBA	90
1	pankaj	BCA	40
2	sudhir	M.Tech	80
3	Geeku	MBA	98

Creating a dataframe

- Create a dataframe with passing rows and columns indexes

```
# Pandas DataFrame with
# row and column indexes.

frame3 = pd.DataFrame(np.arange(16).reshape((4,4)),
         index=['red','blue','yellow','white'],
         columns=['ball','pen','pencil','paper'])
frame3
```

	ball	pen	pencil	paper
red	0	1	2	3
blue	4	5	6	7
yellow	8	9	10	11
white	12	13	14	15

np.arange(16).reshape((4,4)) creates quickly and easily a matrix of values.

4x4 matrix of increasing numbers from 0 to 15.

Dealing with Columns in DataFrames

- To select a column in Pandas DataFrame, access the columns by calling them

```
# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
df[['Name', 'Qualification']]
```

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

Dealing with Columns in DataFrames

- To **add a column** in Pandas DataFrame, we can declare a new list as a column and add to a existing Dataframe

```
# Declare a list that is to be converted into a column
address = ['Delhi', 'Bangalore', 'Chennai', 'Patna']

# Using 'Address' as the column name
# and equating it to the list
df['Address'] = address

# Observe the result
df
```

	Name	Height	Qualification	Address
0	Jai	5.1	Msc	Delhi
1	Princi	6.2	MA	Bangalore
2	Gaurav	5.1	Msc	Chennai
3	Anuj	5.2	Msc	Patna

Dealing with Columns in DataFrames

- To **delete a column** in Pandas DataFrame, we can use the drop()method.
  - Columns is deleted by dropping columns with column names.

```
# dropping passed columns
df.drop(["Address"], axis = 1, inplace = True)

# display
df
```

Dealing with rows in DataFrames

- To retrieve a particular row using its index
  - Use loc() function with passing the target index

```
x = df.loc[2]
x
```

```
Name                    Gaurav
Height                     5.1
Qualification              Msc
Address                Chennai
Name: 2, dtype: object
```

Dealing with rows in DataFrames

- To retrieve a particular row using a values of one attribute

```
result = df.loc[df['Qualification'] == "MA"]
result
```

	Name	Height	Qualification	Address
1	Princi	6.2	MA	Bangalore

Dealing with rows in DataFrames

- To append a new record to a DataFrame

```
new_rec =['Sarah',  5.2,  'PhD', 'Paris']
df.loc[7]=new_rec
df
```

	Name	Height	Qualification	Address
0	Jai	5.1	Msc	Delhi
1	Princi	6.2	MA	Bangalore
2	Gaurav	5.1	Msc	Chennai
3	Anuj	5.2	Msc	Patna
7	Sarah	5.2	PhD	Paris

Dealing with rows in DataFrames

- To drop a particular record from a DataFrame using its index

```
df = df.drop(7)
df
```

	Name	Height	Qualification	Address
0	Jai	5.1	Msc	Delhi
1	Princi	6.2	MA	Bangalore
2	Gaurav	5.1	Msc	Chennai
3	Anuj	5.2	Msc	Patna

Hands-on exercises