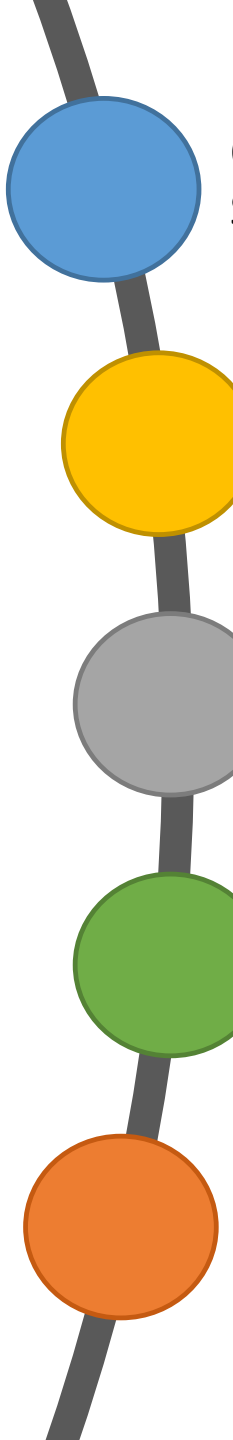
Python for Data Science

Prof. Pradeep Barde

Data science is an interdisciplinary field that uses statistics, scientific computing, scientific methods, processing, scientific visualization, algorithms and systems to extract or extrapolate(predict) knowledge from potentially noisy, structured, or unstructured data.



Overview of Python Libraries for Data Scientists

Reading Data; Selecting and Filtering the Data; Data manipulation, sorting, grouping, rearranging

Plotting the data

Descriptive statistics

Inferential statistics

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

Visualization libraries

- matplotlib
- Seaborn

and many more …

Python Libraries for Data Science

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
- many other python libraries are built on NumPy

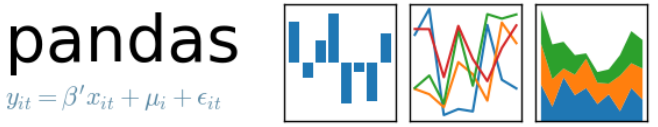
Link: http://www.numpy.org/

Python Libraries for Data Science

SciPy:

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more
- part of SciPy Stack
- built on NumPy

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



Python Libraries for Data Science

Pandas:

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)
- provides tools for data manipulation and analysis
- allows handling missing data

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

Python Libraries for Data Science

SciKit-Learn:

- provides machine learning algorithms: classification, regression, clustering, model validation etc.

- built on NumPy, SciPy and matplotlib

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

Python Libraries for Data Science

matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Link: https://matplotlib.org/

Python Libraries for Data Science

Seaborn:

- based on matplotlib
- provides high level interface for drawing attractive statistical graphics
- Similar (in style) to the popular ggplot2 library in R

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

Start Jupyter nootebook

```
# On the Shared Computing Cluster
[scc1 ~] jupyter notebook
```



Loading Python Libraries

In [ ]:
```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

Press `Shift+Enter` to execute the *jupyter* cell

Data Frame data types

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Data Frame data types

```
In [4]: #Check a particular column type
        df['salary'].dtype
```

```
Out[4]: dtype('int64')
```

```
In [5]: #Check types for all the columns
        df.dtypes
```

```
Out[4]: rank          object
        discipline    object
        phd           int64
        service       int64
        sex           object
        salary        int64
        dtype: object
```

Data Frames attributes

Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Data Frames methods

Unlike attributes, python methods have *parenthesis.*
All attributes and methods can be listed with a *dir()* function: `dir(df)`

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Data Frame: filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the salary value is greater than $120K:

In [ ]:
```
#Calculate mean salary for each professor rank:
df_sub = df[ df['salary'] > 120000 ]
```

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

In [ ]:
```
#Select only those rows that contain female professors:
df_f = df[ df['sex'] == 'Female' ]
```

Data Frames: Slicing

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

Data Frames: Slicing

When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

In [ ]:
```
#Select column salary:
df['salary']
```

When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

In [ ]:
```
#Select column salary:
df[['rank','salary']]
```

Missing Values

There are a number of methods to deal with missing values in the data frame:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Basic statistical Analysis

statsmodel and scikit-learn - both have a number of function for statistical analysis

The first one is mostly used for regular analysis using R style formulas, while scikit-learn is more tailored for Machine Learning.

statsmodels:

- linear regressions
- ANOVA tests
- hypothesis testings
- many more ...

scikit-learn:

- kmeans
- support vector machines
- random forests
- many more ...

See examples in the Tutorial Notebook

Introduction to Pandas

- Provides high-performance, easy-to-use data structures and analysis tools for the Python programming language
- Open-source Python library providing high-performance data manipulation and analysis tool using its powerful data structures
- Name pandas is derived from the word Panel Data – an econometrics term for multidimensional data

Numeric types

- Pandas and base Python uses different names for data types

Python data type	Pandas data type	Description
int	int64	Numeric characters
float	float64	Numeric characters with decimals

  - '64' simply refers to the memory allocated to store data in each cell effectively relates to how many digits it can store in each "cell"
  - 64 bits is equivalent to 8 bytes

Checking data types of each column

dtypes returns a series with the data type of each column

Syntax: **DataFrame.dtypes**

```
cars_data1.dtypes
```

```
Out[37]:
Price          int64
Age          float64
KM            object
FuelType      object
HP            object
MetColor     float64
Automatic      int64
CC             int64
Doors         object
Weight         int64
```

Concise summary of dataframe

`info()` returns a concise summary of a dataframe

- data type of index
- data type of columns
- count of non-null values
- memory usage

Syntax: `DataFrame.info()`

```
cars_data1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1436 entries, 0 to 1435
Data columns (total 10 columns):
Price          1436 non-null int64
Age            1336 non-null float64
KM             1436 non-null object
FuelType       1336 non-null object
HP             1436 non-null object
MetColor       1286 non-null float64
Automatic      1436 non-null int64
CC             1436 non-null int64
Doors          1436 non-null object
Weight         1436 non-null int64
dtypes: float64(2), int64(4), object(4)
memory usage: 163.4+ KB
```

Unique elements of columns

`unique()` is used to find the unique elements of a column

Syntax: `numpy.unique(array)`

```
print(np.unique(cars_data1['KM']))
```

Variable 'Automatic' :

```
print(np.unique(cars_data1['Automatic']))
[0 1]
```

- 'Automatic' has been read as int64 since it has values 0 & 1

Checking unique values of variable 'Doors' :

```
print(np.unique(cars_data['Doors']))
['2' '3' '4' '5' 'five' 'four' 'three']
```

- `replace()` is used to replace a value with the desired value
- Syntax: `DataFrame.replace([to_replace, value, …])`

```
cars_data['Doors'].replace('three',3,inplace=True)
cars_data['Doors'].replace('four',4,inplace=True)
cars_data['Doors'].replace('five',5,inplace=True)
```

Converting variable's data types

`astype()` method is used to explicitly convert data types from one to another

Syntax: `DataFrame.astype(dtype)`

Converting 'MetColor' ,'Automatic' to object data type:

```
cars_data['MetColor'] = cars_data['MetColor'].astype('object')

cars_data['Automatic']=cars_data['Automatic'].astype('object')
```

Data Visualization

- Data visualization allows us to quickly interpret the data and adjust different variables to see their effect
- Technology is increasingly making it easier for us to do so

Why visualize data?

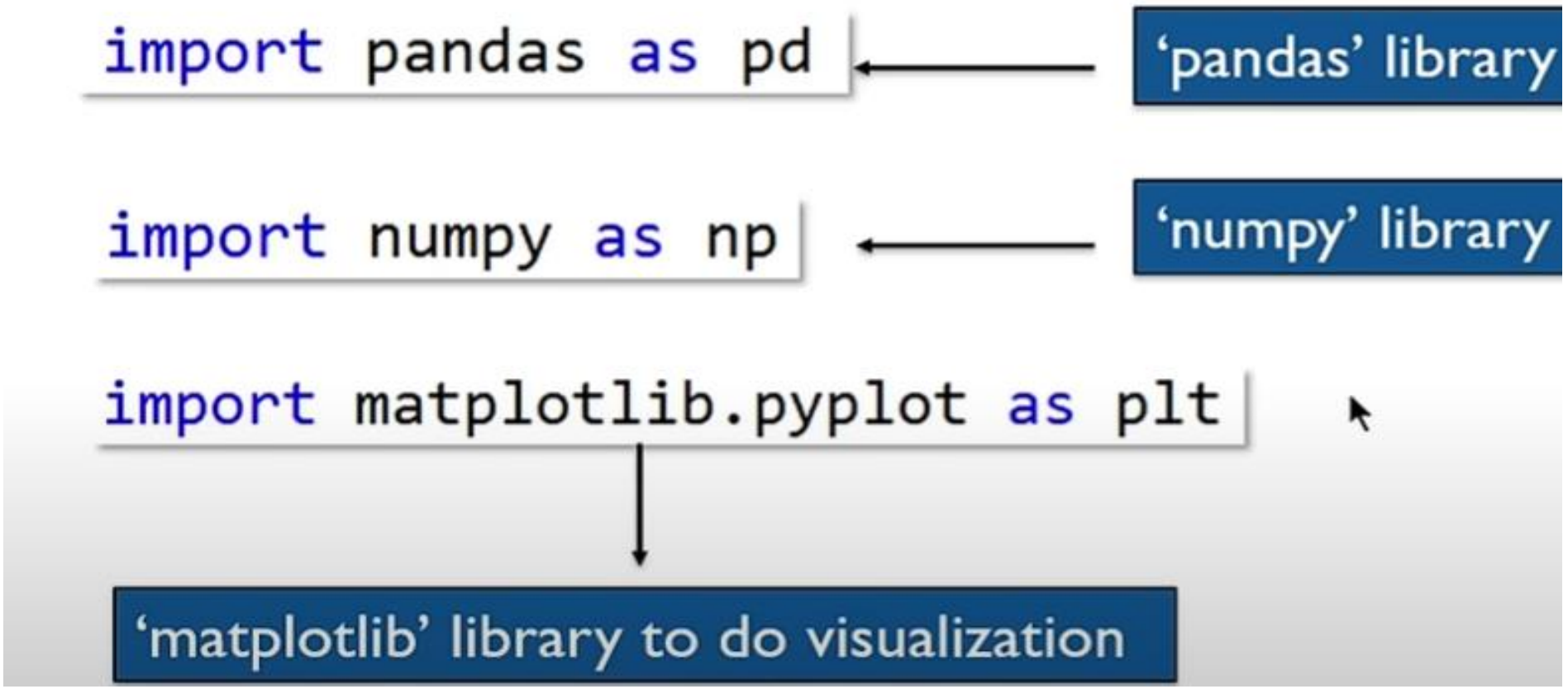
- Observe the patterns
- Identify extreme values that could be anomalies
- Easy interpretation

Popular plotting libraries in Python

Python offers multiple graphing libraries that offers diverse features

• ***matplotlib***	• to create 2D graphs and plots
• ***pandas visualization***	• easy to use interface, built on Matplotlib
• ***seaborn***	• provides a high-level interface for drawing attractive and informative statistical graphics
• ***ggplot***	• based on R's ggplot2, uses Grammar of Graphics
• ***plotly***	• can create interactive plots

- Importing necessary libraries



Using Seaborn library

- The **Seaborn** library makes it super easy to create beautiful and informative visualizations,
- Importing libraries:

```
import seaborn as sns
import matplotlib.pyplot as plt
```

- Count of Male vs Female Passengers:
- sns.countplot(x='sex', data=titanic)
- plt.title('Gender Distribution')
- plt.show()

Identifying missing values

- In Pandas dataframes, missing data is represented by NaN (an acronym for Not a Number)

Identifying missing values

Dataframe.isna.sum(), Dataframe.isnull.sum()

- To check the count of missing values present in each column

```
cars_data2.isna().sum()  (or)  cars_data2.isnull().sum()
```

```
Out[38]:
Price          0
Age          100
KM            15
FuelType     100
HP             6
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
dtype: int64
```

Imputing missing values of 'Age'

- Calculating the mean value of the **Age** variable

```
cars_data2['Age'].mean()
```

```
Out[11]: 55.67215568862275
```

- To fill NA/NaN values using the specified value

DataFrame.fillna()

```
cars_data2['Age'].fillna(cars_data2['Age'].mean(),\
            inplace = True)
```

To detect missing values

To check the count of missing values present in each column `Dataframe.isnull.sum()` is used

```
cars_data.isnull().sum()
```

```
Out[108]:
Price          0
Age          100
KM            15
FuelType     100
HP             6
MetColor     150
Automatic      0
CC             0
Doors          0
Weight         0
dtype: int64
```

Thank You