Nadar Saraswathi College of Engineering & Technology

DATA SCIENCE LAB MANUAL

(Under Revision)

Prepared & Consolidated

by

Vignesh L S

CS3362 DATA SCIENCE LABORATORY (Under Revision)

COURSE OBJECTIVES:

- To understand the python libraries for data science
- To understand the basic Statistical and Probability measures for data science.
- To learn descriptive analytics on the benchmark data sets.
- To apply correlation and regression analytics on standard data sets.
- To present and interpret data using visualization packages in Python.

LIST OF EXPERIMENTS:

1. Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages.

2. Working with Numpy arrays

3. Working with Pandas data frames

4. Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.

5. Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following:

- Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.
- Bivariate analysis: Linear and logistic regression modeling
- Multiple Regression analysis
- Also compare the results of the above analysis for the two data sets.

6. Apply and explore various plotting functions on UCI data sets.

- Normal curves
- Density and contour plots
- Correlation and scatter plots
- Histograms
- Three dimensional plotting

7. Visualizing Geographic Data with Basemap

LIST OF EQUIPMENTS :(30 Students per Batch)

Tools: Python, Numpy, Scipy, Matplotlib, Pandas, statmodels, seaborn, plotly, bokeh

Note: Example data sets like: UCI, Iris, Pima Indians Diabetes etc.

COURSE OUTCOMES:

At the end of this course, the students will be able to:

- CO1: Make use of the python libraries for data science
- CO2: Make use of the basic Statistical and Probability measures for data science.
- CO3: Perform descriptive analytics on the benchmark data sets.
- CO4: Perform correlation and regression analytics on standard data sets
- CO5: Present and interpret data using visualization packages in Python.

1. Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages.

Aim

To Download and install python and its packages using pip installation

Procedure

Install Python Data Science Packages

Python is a high-level and general-purpose programming language with data science and machine learning packages. Use the video below to install on Windows, MacOS, or Linux. As a first step, install Python for Windows, MacOS, or Linux.

Python Packages

The power of Python is in the packages that are available either through the pip or conda package managers. This page is an overview of some of the best packages for machine learning and data science and how to install them.

We will explore the Python packages that are commonly used for data science and machine learning. You may need to install the packages from the terminal, Anaconda prompt, command prompt, or from the Jupyter Notebook. If you have multiple versions of Python or have specific dependencies then use an environment manager such as pyenv. For most users, a single installation is typically sufficient. The Python package manager pip has all of the packages (such as gekko) that we need for this course. If there is an administrative access error, install to the local profile with the --user flag.

```
pip install gekko
```

Gekko

Gekko provides an interface to gradient-based solvers for machine learning and optimization of mixed-integer, differential algebraic equations, and time series models. Gekko provides exact first and second derivatives through automatic differentiation and discretization with simultaneous or sequential methods.

```
pip install gekko
```

Keras

Keras provides an interface for artificial neural networks. Keras acts as an interface for the TensorFlow library. Other backend packages were supported until version 2.4. TensorFlow is now the only backend and is installed separately with pip install tensorflow.

```
pip install keras
```

Matplotlib

The package matplotlib generates plots in Python.

```
pip install matplotlib
```

Numpy

Numpy is a numerical computing package for mathematics, science, and engineering. Many data science packages use Numpy as a dependency.

```
pip install numpy
```

OpenCV

OpenCV (Open Source Computer Vision Library) is a package for real-time computer vision and developed with support from Intel Research.

```
pip install opencv-python
```

Pandas

Pandas visualizes and manipulates data tables. There are many functions that allow efficient manipulation for the preliminary steps of data analysis problems.

```
pip install pandas
```

Plotly

Plotly renders interactive plots with HTML and JavaScript. Plotly Express is included with Plotly.

```
pip install plotly
```

PyTorch

PyTorch enables deep learning, computer vision, and natural language processing. Development is led by Facebook's AI Research lab (FAIR).

```
pip install torch
```

Scikit-Learn

Scikit-Learn (or sklearn) includes a wide variety of classification, regression and clustering algorithms including neural network, support vector machine, random forest, gradient boosting, k-means clustering, and other supervised or unsupervised learning methods.

```
pip install scikit-learn
```

SciPy

SciPy is a general-purpose package for mathematics, science, and engineering and extends the base capabilities of NumPy.

```
pip install scipy
```

Seaborn

Seaborn is built on matplotlib, and produces detailed plots in few lines of code.

```
pip install seaborn
```

Statsmodels

Statsmodels is a package for exploring data, estimating statistical models, and performing statistical tests. It include descriptive statistics, statistical tests, plotting functions, and result statistics.

```
pip install statsmodels
```

TensorFlow

TensorFlow is an open source machine learning platform with particular focus on training and inference of deep neural networks. Development is led by the Google Brain team.

```
pip install tensorflow
```

Working with Numpy arrays

CREATE A NUMPY NDARRAY OBJECT

NumPy is used to work with arrays. The array object in NumPy is called ndarray.

Example

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)

print(type(arr))
```

To create an ndarray, we can pass a list, tuple or any array-like object into the array() method, and it will be converted into an ndarray:

Example

Use a tuple to create a NumPy array:

```
import numpy as np

arr = np.array((1, 2, 3, 4, 5))

print(arr)
```

Dimensions in Arrays

A dimension in arrays is one level of array depth (nested arrays).

0-D Arrays

0-D arrays, or Scalars, are the elements in an array. Each value in an array is a 0-D array.

Example

Create a 0-D array with value 42

```
import numpy as np

arr = np.array(42)

print(arr)
```

1-D Arrays

An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.

These are the most common and basic arrays.

Example

Create a 1-D array containing the values 1,2,3,4,5:

```
import numpy as np

arr = np.array([1, 2, 3, 4, 5])

print(arr)
```

2-D Arrays

An array that has 1-D arrays as its elements is called a 2-D array. These are often used to represent matrix or 2nd order tensors.

Create a 2-D array containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

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

print(arr)
```

3-D arrays

An array that has 2-D arrays (matrices) as its elements is called 3-D array.

These are often used to represent a 3rd order tensor.

Example

Create a 3-D array with two 2-D arrays, both containing two arrays with the values 1,2,3 and 4,5,6:

```
import numpy as np

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

print(arr)
```

Check Number of Dimensions?

NumPy Arrays provides the ndim attribute that returns an integer that tells us how many dimensions the array have.

Example

Check how many dimensions the arrays have:

```
import numpy as np

a = np.array(42)

b = np.array([1, 2, 3, 4, 5])

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

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

print(a.ndim)
print(b.ndim)

print(c.ndim)

print(d.ndim)
```

Higher Dimensional Arrays

An array can have any number of dimensions.

When the array is created, you can define the number of dimensions by using the ndmin argument.

Example

Create an array with 5 dimensions and verify that it has 5 dimensions:

```
import numpy as np

arr = np.array([1, 2, 3, 4], ndmin=5)

print(arr)

print('number of dimensions :', arr.ndim)
```

In this array the innermost dimension (5th dim) has 4 elements, the 4th dim has 1 element that is the vector, the 3rd dim has 1 element that is the matrix with the vector, the 2nd dim has 1 element that is 3D array and 1st dim has 1 element that is a 4D array.

Working with Pandas data frames

A Pandas DataFrame is a 2 dimensional data structure, like a 2 dimensional array, or a table with rows and columns.

Example

Create a simple Pandas DataFrame:

```
import pandas as pd

data = {

  "calories": [420, 380, 390],

  "duration": [50, 40, 45]

}
```

#load data into a DataFrame object:

```
df = pd.DataFrame(data)

print(df)
```

Result

```
   calories  duration

 0     420       50

 1     380       40

 2     390       45
```

Locate Row

As you can see from the result above, the DataFrame is like a table with rows and columns.

Pandas use the loc attribute to return one or more specified row(s)

Example

Return row 0:

#refer to the row index:

```
print(df.loc[0])
```

Result

```
calories    420
duration     50
Name: 0, dtype: int64
```

Note: This example returns a Pandas Series.

Example

Return row 0 and 1:

#use a list of indexes:

```
print(df.loc[[0, 1]])
```

Result

```
   calories  duration
0       420        50
1       380        40
```

Note: When using [], the result is a Pandas DataFrame.

Named Indexes

With the index argument, you can name your own indexes.

Example

Add a list of names to give each row a name:

```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
df = pd.DataFrame(data, index = ["day1", "day2", "day3"])
print(df)
```

Result

```
      calories  duration
day1      420        50
day2      380        40
day3      390        45
```

Locate Named Indexes

Use the named index in the loc attribute to return the specified row(s).

Example

Return "day2":

#refer to the named index:

```
print(df.loc["day2"])
```

Result

```
calories    380
duration     40
Name: 0, dtype: int64
```

Load Files Into a DataFrame

If your data sets are stored in a file, Pandas can load them into a DataFrame.

Example

Load a comma separated file (CSV file) into a DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df)
```