**Applied Machine Learning**

**Laboratory Manual**

# LIST OF EXPERIMENTS

Course Code: 3171617

Course Title: Applied Machine Learning

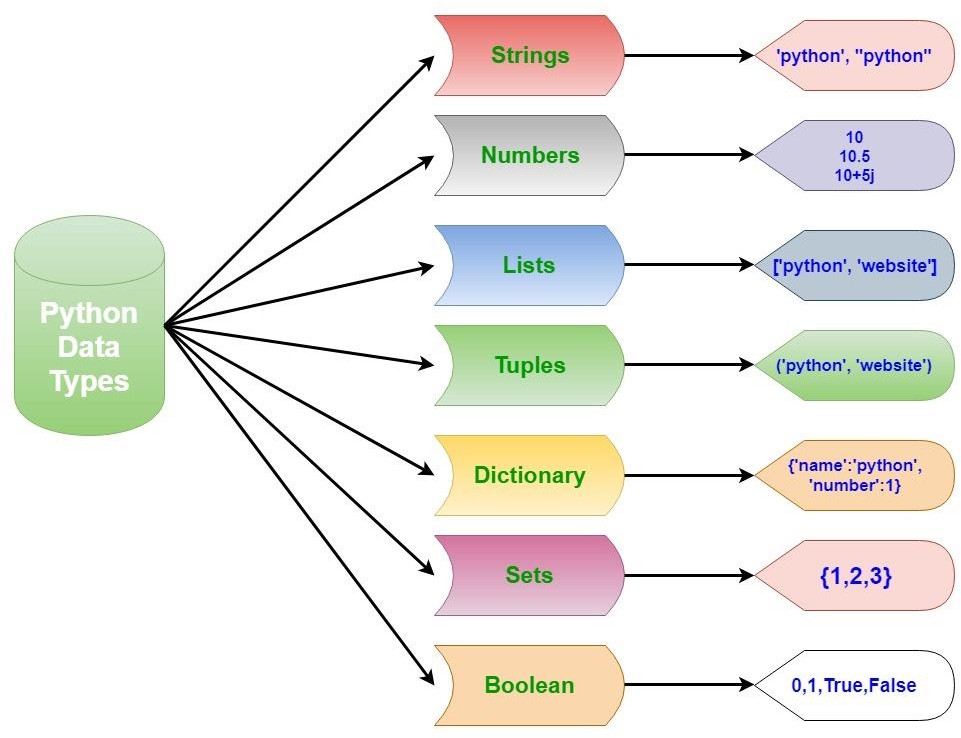
|  |  |
| --- | --- |
| **Sr.No** | **Name of the Experiment** |
| 1. | Write a python program to compute   * Central Tendency Measures: Mean, Median, Mode * Measure of Dispersion: Variance, Standard Deviation |
| 2. | Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy |
| 3. | Study of Python Libraries for ML application such as Pandas and Matplotlib |
| 4. | Study of Python Libraries for Multiple Linear Regression. |
| 5. | Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample. |
| 6. | Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets. |
| 7. | Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API. |
| 8. | Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem. |

**Experiment No: 1**

**Write a python program to compute**

* + **Central tendency measures: Mean, Median, Mode**
  + **Measure of Dispersion: Variance, Standard Deviation**

***Python Data Types**



* **Numeric Types:**
  1. **Integers:**

In Python 3, there is effectively no limit to how long an integer value can be. Of course, it is constrained by the amount of memory your system has.

>>> print(10) 10

>>> type(10)

<class 'int'>

* 1. **Floating Point Numbers:**

The float type in Python designates a floating-point number. float values are specified with a decimal point. Optionally, the character e or E followed by a positive or negative integer may be appended to specify scientific notation

>>> 4.2

4.2

>>> .4e7 4000000.0

>>> type(.4e7)

<class 'float'>

>>> 4.2e-4 0.00042

* 1. **Complex Numbers**

Complex numbers are specified as <real part>+<imaginary part>j.

>>> 2+3j (2+3j)

>>> type(2+3j)

<class 'complex'>

### Strings:

Strings are sequences of character data. The string type in Python is called str. String literals may be delimited using either single or double quotes. All the characters between the opening delimiter and matching closing delimiter are part of the string.

A string in Python can contain as many characters as you wish. The only limit is your machine‘s memory resources. A string can also be empty.

>>> print("I am a string.") I am a string.

>>> type("I am a string.")

<class 'str'>

>>> ''

''

A raw string literal is preceded by r or R, which specifies that escape sequences in the associated string are not translated. The backslash character is left in the string.

>>> print('foo\nbar') foo

bar

>>> print(r'foo\nbar') foo\nbar

>>> print('foo\\bar') foo\bar

>>> print(R'foo\\bar')

foo\\bar

### Boolean Type:

Python 3 provides a Boolean data type. Objects of Boolean type may have one of two values, True or False.

>>> type(True)

<class 'bool'>

>>> type(False)

<class 'bool'>

### Python List:

List is an ordered sequence of items. It is one of the most used datatype in Python and is very flexible. All the items in a list do not need to be of the same type. Declaring a list is pretty straight forward. Items separated by commas are enclosed within brackets [ ]. We can use the slicing operator [ ] to extract an item or a range of items from a list. Index starts form 0 in Python.

Lists are mutable, meaning, value of elements of a list can be altered.

>>> a = [1, 2.2, 'python']

### Python Tuple:

Tuple is an ordered sequence of items same as list. The only difference is that tuples are immutable. Tuples once created cannot be modified. Tuples are used to write-protect data and are usually faster than list as it cannot change dynamically. It is defined within parentheses () where items are separated by commas. We can use the slicing operator [] to extract items but we cannot change its value.

>>> t = (5,'program', 1+3j)

### Python Set:

Set is an unordered collection of unique items. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered. We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates. Since, set are unordered collection, indexing has no meaning. Hence the slicing operator [] does not work.

|  |  |  |  |
| --- | --- | --- | --- |
| >>> a = {1,2,2,3,3,3} | | |  |
| >>> a |  | | |
| {1, 2, 3} | |  | |

### Python Dictionary:

Dictionary is an unordered collection of key-value pairs. It is generally used when we have a huge amount of data. Dictionaries are optimized for retrieving data. We must know the key to retrieve the value. In Python, dictionaries are defined within braces {} with each item being a pair in the form key:value. Key and value can be of any type. We use key to retrieve the respective value. But not the other way around.

|  |  |  |  |
| --- | --- | --- | --- |
| >>> d = {1:'value','key':2} | | |  |
| >>> type(d) |  | | |
| <class 'dict'> | |  | |

### *Operators in Python

Operators are used to perform operations on variables and values. Python divides the operators in the following groups:

1. **Arithmetic operators**

Arithmetic operators are used to perform mathematical operations like addition, subtraction, multiplication and division.

|  |  |  |  |
| --- | --- | --- | --- |
| + | Addition: adds two operands x + y |  | |
| - | Subtraction: subtracts two operands | x - y | |
| \* | Multiplication: multiplies two operands x \* y | |  |
| / | Division (float): divides the first operand by the second | | x / y |
| // | Division (floor): divides the first operand by the second | | x // y |

% Modulus: returns the remainder when first operand is divided by the second x % y

1. **Comparison/Relational operators**

Relational operators compares the values. It either returns True or False according to the condition.

> Greater than: True if left operand is greater than the right x > y

< Less than: True if left operand is less than the right x < y

== Equal to: True if both operands are equal x == y

!= Not equal to - True if operands are not equal x != y

>= Greater than or equal to: True if left operand is greater than or equal to the right x >= y

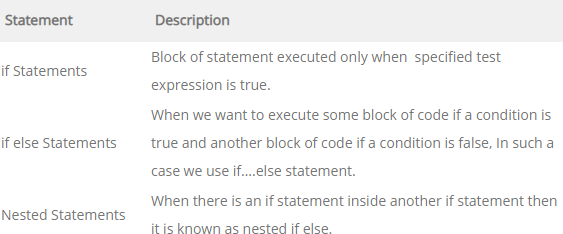
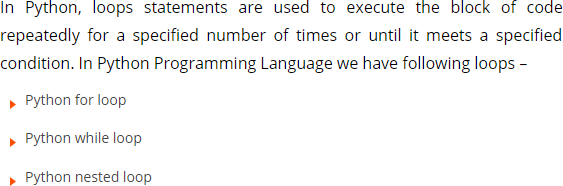
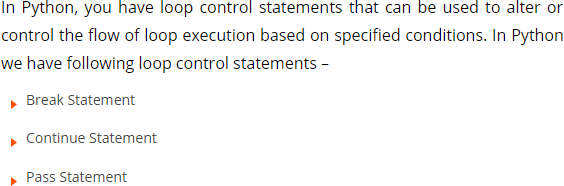
<= Less than or equal to: True if left operand is less than or equal to the right x<= y

1. **Logical operators**

Logical operators perform Logical AND, Logical OR and Logical NOT operations.

|  |  |  |
| --- | --- | --- |
| and | Logical AND: True if both the operands are true | x and y |
| or | Logical OR: True if either of the operands is true | x or y |
| not | Logical NOT: True if operand is false | not x |

***Control Statements in Python**

1. **Python Decision Making Statements**
2. **Python Loops Statements**
3. **Loop Control Statements**

***Executing Python Program**

This describes the environment in which Python programs are executed. This describes the runtime behavior of the interpreter, including program startup, configuration, and program termination.

* **Anaconda Navigator – Jupyter Notebook**

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, machine learning and data science that aims to simplify package management and deployment.

The notebook extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results. A notebook kernel is a ―computational engine‖ that executes the code contained in a Notebook document. The ipython kernel, referenced in this guide, executes python code.

The Jupyter notebook combines two components:

* A web application: a browser-based tool for interactive authoring of documents which combine explanatory text, mathematics, computations and their rich media output.
* Notebook documents: a representation of all content visible in the web application, including inputs and outputs of the computations, explanatory text, mathematics, images, and rich media representations of objects.
* **PyCharm IDE**

PyCharm is an integrated development environment (IDE) used in computer programming, specifically for the Python language. It is developed by the JetBrains. It provides code analysis, a graphical debugger, an integrated unit tester, integration with version control systems, and supports web development with Django as well as Data Science with Anaconda.

* **Google COLAB**

Colaboratory is a research tool for machine learning education and research. It‘s a Jupyter notebook environment that requires no setup to use. Google Colab is a free cloud service and now it supports free GPU! You can: improve your Python programming language coding skills. To start working with Colab you first need to log in to your google account, then go to this link [https://colab.research.google.com](https://colab.research.google.com/).

### *Central Tendency Measures

A measure of central tendency (also referred to as measures of centre or central location) is a summary measure that attempts to describe a whole set of data with a single value that represents the middle or centre of its distribution.

There are three main measures of central tendency: the mode, the median and the mean. Each of these measures describes a different indication of the typical or central value in the distribution.

1. **Mean**

The mean is the sum of the value of each observation in a dataset divided by the number of observations. This is also known as the arithmetic average. Looking at the retirement age distribution again:

54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

The mean is calculated by adding together all the values (54+54+54+55+56+57+57+58+58+60+60 = 623) and dividing by the number of observations

(11) which equals 56.6 years.

1. **Median**

The median is the middle value in distribution when the values are arranged in ascending or descending order.

In a distribution with an odd number of observations, the median value is the middle value.

54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

Looking at the retirement age distribution (which has 11 observations), the median is the middle value, which is 57 years.

When the distribution has an even number of observations, the median value is the mean of the two middle values. In the following distribution,

52, 54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

The middle two values are 56 and 57; therefore the median equals 56.5 years.

1. **Mode**

The mode is the most commonly occurring value in a distribution. Consider this dataset showing the retirement age of 11 people, in whole years:

54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

The most commonly occurring value is 54, therefore the mode of this distribution is 54 years.

### * Measure of Dispersion

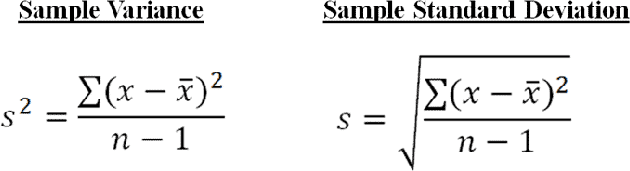
Measures of spread describe how similar or varied the set of observed values are for a particular variable (data item). Measures of spread include the range, quartiles and the interquartile range, variance and standard deviation.

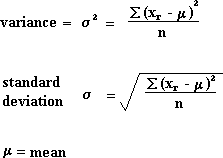
The spread of the values can be measured for quantitative data, as the variables are numeric and can be arranged into a logical order with a low end value and a high end value.

* **Variance and Standard Deviation**

The variance and the standard deviation are measures of the spread of the data around the mean. They summarise how close each observed data value is to the mean value.

In datasets with a small spread all values are very close to the mean, resulting in a small variance and standard deviation. Where a dataset is more dispersed, values are spread further away from the mean, leading to a larger variance and standard deviation.

The smaller the variance and standard deviation, the more the mean value is indicative of the whole dataset. Therefore, if all values of a dataset are the same, the standard deviation and variance are zero.



## Experiment No: 2

**Study of Python Basic Libraries such as Statistics, Math, Numpy and Scipy**

* **Python Libraries**

There are a lot of reasons why Python is popular among developers and one of them is that it has an amazingly large collection of libraries that users can work with. In this Python Library, we will discuss Python Standard library and different libraries offered by Python Programming Language: scipy, numpy, etc.

We know that a module is a file with some Python code, and a package is a directory for sub packages and modules. A Python library is a reusable chunk of code that you may want to include in your programs/ projects. Here, a ‗library‘ loosely describes a collection of core modules. Essentially, then, a library is a collection of modules. A package is a library that can be installed using a package manager like npm.

* **Python Standard Library**

The Python Standard Library is a collection of script modules accessible to a Python program to simplify the programming process and removing the need to rewrite commonly used commands. They can be used by 'calling/importing' them at the beginning of a script. A list of the Standard Library modules that are most important

* time
* sys
* csv
* math
* random
* pip
* os
* statistics
* tkinter
* socket

To display a list of all available modules, use the following command in the Python console:

>>> help('modules')

### *List of important Python Libraries

* Python Libraries for Data Collection
  + Beautiful Soup
  + Scrapy
  + Selenium
* Python Libraries for Data Cleaning and Manipulation
  + Pandas
  + PyOD
  + NumPy
  + Scipy
  + Spacy
* Python Libraries for Data Visualization
  + Matplotlib
  + Seaborn
  + Bokeh
* Python Libraries for Modeling
  + Scikit-learn
  + TensorFlow
  + PyTorch
* Python Libraries for Model Interpretability
  + Lime
  + H2O
* Python Libraries for Audio Processing
  + Librosa
  + Madmom
  + pyAudioAnalysis
* Python Libraries for Image Processing
  + OpenCV-Python
  + Scikit-image
  + Pillow
* Python Libraries for Database
  + Psycopg
  + SQLAlchemy
* Python Libraries for Deployment
  + Flask

*How to install additional python libraries such as Numpy, Scipy, Pandas and Matplotlib

* In Anaconda Environment: use these commands Conda install numpy scipy pandas matplotlib
* In Windows system using command prompt

Python –m pip install numpy scipy pandas matplotlib

***Python Math Library**

The math module is a standard module in Python and is always available. To use mathematical functions under this module, you have to import the module using import math. It gives access to the underlying C library functions. This module does not support complex datatypes. The cmath module is the complex counterpart.

|  |  |
| --- | --- |
| List of Functions in Python Math Module | |
| **Function** | **Description** |
| ceil(x) | Returns the smallest integer greater than or equal to x. |
| copysign(x, y) | Returns x with the sign of y |
| fabs(x) | Returns the absolute value of x |
| factorial(x) | Returns the factorial of x |
| floor(x) | Returns the largest integer less than or equal to x |
| fmod(x, y) | Returns the remainder when x is divided by y |
| frexp(x) | Returns the mantissa and exponent of x as the pair (m, e) |
| fsum(iterable) | Returns an accurate floating point sum of values in the iterable |
| isfinite(x) | Returns True if x is neither an infinity nor a NaN (Not a Number) |
| isinf(x) | Returns True if x is a positive or negative infinity |
| isnan(x) | Returns True if x is a NaN |
| ldexp(x, i) | Returns x \* (2\*\*i) |
| modf(x) | Returns the fractional and integer parts of x |
| trunc(x) | Returns the truncated integer value of x |
| exp(x) | Returns e\*\*x |
| expm1(x) | Returns e\*\*x - 1 |
| log(x[, base]) | Returns the logarithm of x to the base (defaults to e) |
| log1p(x) | Returns the natural logarithm of 1+x |
| log2(x) | Returns the base-2 logarithm of x |
| log10(x) | Returns the base-10 logarithm of x |
| pow(x, y) | Returns x raised to the power y |
| sqrt(x) | Returns the square root of x |
| acos(x) | Returns the arc cosine of x |
| asin(x) | Returns the arc sine of x |

|  |  |
| --- | --- |
| atan(x) | Returns the arc tangent of x |
| atan2(y, x) | Returns atan(y / x) |
| cos(x) | Returns the cosine of x |
| hypot(x, y) | Returns the Euclidean norm, sqrt(x\*x + y\*y) |
| sin(x) | Returns the sine of x |
| tan(x) | Returns the tangent of x |
| degrees(x) | Converts angle x from radians to degrees |
| radians(x) | Converts angle x from degrees to radians |
| acosh(x) | Returns the inverse hyperbolic cosine of x |
| asinh(x) | Returns the inverse hyperbolic sine of x |
| atanh(x) | Returns the inverse hyperbolic tangent of x |
| cosh(x) | Returns the hyperbolic cosine of x |
| sinh(x) | Returns the hyperbolic cosine of x |
| tanh(x) | Returns the hyperbolic tangent of x |
| erf(x) | Returns the error function at x |
| erfc(x) | Returns the complementary error function at x |
| gamma(x) | Returns the Gamma function at x |
| lgamma(x) | Returns the natural logarithm of the absolute value of the Gamma function at x |
| Pi | Mathematical constant, the ratio of circumference of a circle to it's diameter (3.14159...) |
| E | mathematical constant e (2.71828...) |

***Python Statistics library**

This module provides functions for calculating mathematical statistics of numeric (Real-valued) data. The statistics module comes with very useful functions like: Mean, median, mode, standard deviation, and variance.

The four functions we'll use in this post are common in statistics:

1. mean - average value
2. median - middle value
3. mode - most often value
4. standard deviation - spread of values

* **Averages and measures of central location**

These functions calculate an average or typical value from a population or sample. mean() Arithmetic mean (―average‖) of data.

harmonic\_mean() Harmonic mean of data.

median() Median (middle value) of data. median\_low() Low median of data.

median\_high() High median of data.

median\_grouped() Median, or 50th percentile, of grouped data. mode() Mode (most common value) of discrete data.

* **Measures of spread**

These functions calculate a measure of how much the population or sample tends to deviate from the typical or average values.

pstdev() Population standard deviation of data.

pvariance() Population variance of data. stdev() Sample standard deviation of data. variance() Sample variance of data.

***Importance of Numpy and Scipy libraries**

NumPy (Numerical Python) is a linear algebra library in Python. It is a very important library on which almost every data science or machine learning Python packages such as SciPy (Scientific Python), Mat−plotlib (plotting library), Scikit-learn, etc depends on to a reasonable extent.

NumPy is very useful for performing mathematical and logical operations on Arrays. It provides an abundance of useful features for operations on n-arrays and matrices in Python.

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* a powerful N-dimensional array object
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

The SciPy library is one of the core packages that make up the SciPy stack. It provides many user- friendly and efficient numerical routines such as routines for numerical integration, interpolation, optimization, linear algebra and statistics.

***Python Numpy Library**

NumPy is an open source library available in Python that aids in mathematical, scientific, engineering, and data science programming. NumPy is an incredible library to perform mathematical and statistical operations. It works perfectly well for multi-dimensional arrays and matrices multiplication

For any scientific project, NumPy is the tool to know. It has been built to work with the N- dimensional array, linear algebra, random number, Fourier transform, etc. It can be integrated to C/C++ and Fortran.

NumPy is a programming language that deals with multi-dimensional arrays and matrices. On top of the arrays and matrices, NumPy supports a large number of mathematical operations.

NumPy is memory efficiency, meaning it can handle the vast amount of data more accessible than any other library. Besides, NumPy is very convenient to work with, especially for matrix multiplication and reshaping. On top of that, NumPy is fast. In fact, TensorFlow and Scikit learn to use NumPy array to compute the matrix multiplication in the back end.

* **Arrays in NumPy:** NumPy‘s main object is the homogeneous multidimensional array.
* It is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers.
* In NumPy dimensions are called *axes*. The number of axes is *rank*.
* NumPy’s array class is called **ndarray**. It is also known by the alias **array**.

We use python numpy array instead of a list because of the below three reasons:

1. Less Memory
2. Fast
3. Convenient

* **Numpy Functions**

Numpy arrays carry attributes around with them. The most important ones are: ndim: The number of axes or rank of the array

shape: A tuple containing the length in each dimension size: The total number of elements

*import numpy*

*x = numpy.array([[1,2,3], [4,5,6], [7,8,9]]) # 3x3 matrix print(x.ndim) # Prints 2*

*print(x.shape) # Prints (3L, 3L) print(x.size) # Prints 9*

Can be used just like Python lists

x[1] will access the second element x[-1] will access the last element

Arithmetic operations apply element wise *a = numpy.array( [20,30,40,50] ) b = numpy.arange( 4 )*

*c = a-b*

*#c => array([20, 29, 38, 47])*

* **Built-in Methods**

Many standard numerical functions are available as methods out of the box:

*x = numpy.array([1,2,3,4,5]) avg = x.mean()*

*sum = x.sum()*

*sx = numpy.sin(x)*

### Functions and Methods Overview

Here is a list of some useful NumPy functions and methods names ordered in categories.

Array Creation

[arange](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.arange.html#numpy.arange), [array](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.array.html#numpy.array), [copy](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.copy.html#numpy.copy), [empty](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.empty.html#numpy.empty), [empty\_like](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.empty_like.html#numpy.empty_like), [eye](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.eye.html#numpy.eye), [fromfile](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.fromfile.html#numpy.fromfile), [fromfunction](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.fromfunction.html#numpy.fromfunction), [identity](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.identity.html#numpy.identity), [linspace](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linspace.html#numpy.linspace), [logspace](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.logspace.html#numpy.logspace), [mgrid](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.mgrid.html#numpy.mgrid), [ogrid](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ogrid.html#numpy.ogrid), [ones](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ones.html#numpy.ones), [ones\_like](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ones_like.html#numpy.ones_like), *r*, [zeros](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.zeros.html#numpy.zeros), [zeros\_like](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.zeros_like.html#numpy.zeros_like)

Conversions

[ndarray.astype](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ndarray.astype.html#numpy.ndarray.astype), [atleast\_1d](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.atleast_1d.html#numpy.atleast_1d), [atleast\_2d](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.atleast_2d.html#numpy.atleast_2d), [atleast\_3d](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.atleast_3d.html#numpy.atleast_3d), [mat](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.mat.html#numpy.mat)

Manipulations

[array\_split](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.array_split.html#numpy.array_split), [column\_stack](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.column_stack.html#numpy.column_stack), [concatenate](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.concatenate.html#numpy.concatenate), [diagonal](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.diagonal.html#numpy.diagonal), [dsplit](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.dsplit.html#numpy.dsplit), [dstack](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.dstack.html#numpy.dstack), [hsplit](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.hsplit.html#numpy.hsplit), [hstack](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.hstack.html#numpy.hstack), [ndarray.item](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ndarray.item.html#numpy.ndarray.item), [newaxis](https://docs.scipy.org/doc/numpy-1.13.0/reference/arrays.indexing.html#numpy.newaxis), [ravel](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ravel.html#numpy.ravel), [repeat](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.repeat.html#numpy.repeat), [reshape](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.reshape.html#numpy.reshape), [resize](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.resize.html#numpy.resize), [squeeze](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.squeeze.html#numpy.squeeze), [swapaxes](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.swapaxes.html#numpy.swapaxes), [take](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.take.html#numpy.take), [transpose](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.transpose.html#numpy.transpose), [vsplit](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.vsplit.html#numpy.vsplit), [vstack](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.vstack.html#numpy.vstack)

Questions

[all](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.all.html#numpy.all), [any](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.any.html#numpy.any), [nonzero](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.nonzero.html#numpy.nonzero), [where](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.where.html#numpy.where)

Ordering

[argmax](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.argmax.html#numpy.argmax), [argmin](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.argmin.html#numpy.argmin), [argsort](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.argsort.html#numpy.argsort), [max](https://docs.python.org/dev/library/functions.html#max), [min](https://docs.python.org/dev/library/functions.html#min), [ptp](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ptp.html#numpy.ptp), [searchsorted](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.searchsorted.html#numpy.searchsorted), [sort](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.sort.html#numpy.sort)

Operations

[choose](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.choose.html#numpy.choose), [compress](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.compress.html#numpy.compress), [cumprod](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.cumprod.html#numpy.cumprod), [cumsum](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.cumsum.html#numpy.cumsum), [inner](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.inner.html#numpy.inner), [ndarray.fill](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.ndarray.fill.html#numpy.ndarray.fill), [imag](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.imag.html#numpy.imag), [prod](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.prod.html#numpy.prod), [put](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.put.html#numpy.put), [putmask](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.putmask.html#numpy.putmask), [real](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.real.html#numpy.real), [sum](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.sum.html#numpy.sum)

Basic Statistics

[cov](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.cov.html#numpy.cov), [mean](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.mean.html#numpy.mean), [std](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.std.html#numpy.std), [var](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.var.html#numpy.var)

Basic Linear Algebra

[cross](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.cross.html#numpy.cross), [dot](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.dot.html#numpy.dot), [outer](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.outer.html#numpy.outer), [linalg.svd](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.linalg.svd.html#numpy.linalg.svd), [vdot](https://docs.scipy.org/doc/numpy-1.13.0/reference/generated/numpy.vdot.html#numpy.vdot)

***Python Scipy Library**

SciPy is an Open Source Python-based library, which is used in mathematics, scientific computing, Engineering, and technical computing. SciPy also pronounced as "Sigh Pi."

* SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
* SciPy is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
* Easy to use and understand as well as fast computational power.
* It can operate on an array of NumPy library.
* **Numpy VS SciPy Numpy:**

1. Numpy is written in C and use for mathematical or numeric calculation.
2. It is faster than other Python Libraries
3. Numpy is the most useful library for Data Science to perform basic calculations.
4. Numpy contains nothing but array data type which performs the most basic operation like sorting, shaping, indexing, etc.

**SciPy:**

1. SciPy is built in top of the NumPy
2. SciPy is a fully-featured version of Linear Algebra while Numpy contains only a few features.
3. Most new Data Science features are available in Scipy rather than Numpy.

### A concise list of SciPy sub-modules is shown below:

|  |  |
| --- | --- |
| Scipy.fftpack | Fast Fourier Transform |
| Scipy.interpolate | Interpolation |
| Scipy.integrate | Numerical Integration |
| Scipy.linalg | Linear Algebra |
| Scipy.io | File Input/Output |
| Scipy.optimize | Optimization and Fits |
| Scipy.stats | Statistics |
| Scipy.signal | Signal Processing |

* **Linear Algebra with SciPy**
  + Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.
  + Performance of Linear Algebra is very fast compared to BLAS and LAPACK.
  + Linear algebra routine accepts two-dimensional array object and output is also a two- dimensional array.

Now let's do some test with **scipy.linalg,**

Calculating **determinant** of a two-dimensional matrix,

*from scipy import linalg import numpy as np #define square matrix*

*two\_d\_array = np.array([ [4,5], [3,2] ]) #pass values to det() function linalg.det( two\_d\_array )*

* **Eigenvalues and Eigenvector –** scipy.linalg.eig()
  + The most common problem in linear algebra is eigenvalues and eigenvector which can be easily solved using **eig()** function.
  + Now lets we find the Eigenvalue of (**X**) and correspond eigenvector of a two-dimensional square matrix.

Example,

*from scipy import linalg import numpy as np*

*#define two dimensional array arr = np.array([[5,4],[6,3]]) #pass value into function*

*eg\_val, eg\_vect = linalg.eig(arr) #get eigenvalues*

*print(eg\_val) #get eigenvectors print(eg\_vect)*

## Experiment No: 3

**Study of Python Libraries for ML application such as Pandas and Matplotlib**

*List important ML libraries

* + - Python Libraries for Machine Learning
      * Numpy
      * Scipy
      * Scikit-learn
      * Theano
      * TensorFlow
      * Keras
      * PyTorch
      * Pandas
      * Matplotlib

***Importance of Pandas library**

Pandas is an open source, BSD-licensed library providing high-performance, easy-to-use data structures and data analysis tools for the Python programming language.

Pandas makes importing, analyzing, and visualizing data much easier. It builds on packages like NumPy and matplotlib to give you a single, convenient, place to do most of your data analysis and visualization work.

**Advantages of Pandas Library**

There are many benefits of Python Pandas library, listing them all would probably take more time than what it takes to learn the library. Therefore, these are the core advantages of using the Pandas library:

1. Data representation

Pandas provide extremely streamlined forms of data representation. This helps to analyze and understand data better. Simpler data representation facilitates better results for data science projects.

1. Less writing and more work done

It is one of the best advantages of Pandas. What would have taken multiple lines in Python without any support libraries, can simply be achieved through 1-2 lines with the use of Pandas.

Thus, using Pandas helps to shorten the procedure of handling data. With the time saved, we can focus more on data analysis algorithms.

1. An extensive set of features

Pandas are really powerful. They provide you with a huge set of important commands and features which are used to easily analyze your data. We can use Pandas to perform various tasks like filtering your data according to certain conditions, or segmenting and segregating the data according to preference, etc.

1. Efficiently handles large data

Wes McKinney, the creator of Pandas, made the python library to mainly handle large datasets efficiently. Pandas help to save a lot of time by importing large amounts of data very fast.

1. Makes data flexible and customizable

Pandas provide a huge feature set to apply on the data you have so that you can customize, edit and pivot it according to your own will and desire. This helps to bring the most out of your data.

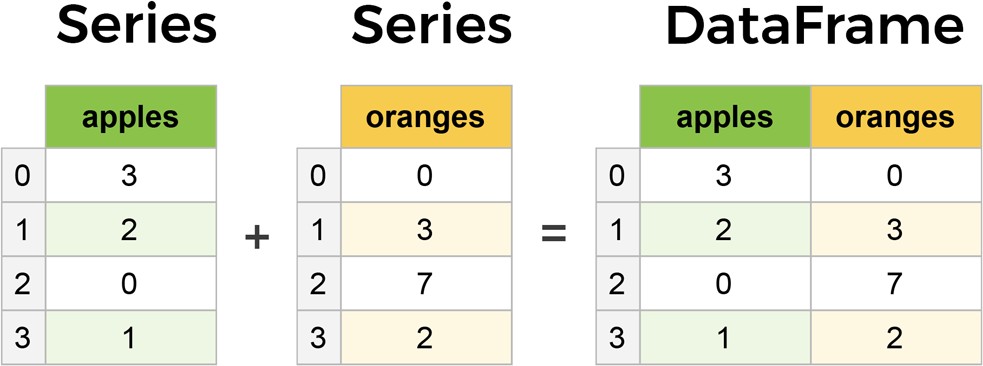
1. Made for Python

Python programming has become one of the most sought after programming languages in the world, with its extensive amount of features and the sheer amount of productivity it provides. Therefore, being able to code Pandas in Python, enables you to tap into the power of the various other features and libraries which will use with Python. Some of these libraries are NumPy, SciPy, MatPlotLib, etc.

***Pandas Library**

The primary two components of pandas are the Series and DataFrame.

A Series is essentially a column, and a DataFrame is a multi-dimensional table made up of a collection of Series.

DataFrames and Series are quite similar in that many operations that you can do with one you can do with the other, such as filling in null values and calculating the mean.

* **Reading data from CSVs**

With CSV files all you need is a single line to load in the data:

*df = pd.read\_csv('purchases.csv') df*

Let's load in the IMDB movies dataset to begin:

*movies\_df = pd.read\_csv("IMDB-Movie-Data.csv", index\_col="Title")*

We're loading this dataset from a CSV and designating the movie titles to be our index.

* **Viewing your data**

The first thing to do when opening a new dataset is print out a few rows to keep as a visual reference. We accomplish this with .head():

*movies\_df.head()*

Another fast and useful attribute is .shape, which outputs just a tuple of (rows, columns):

*movies\_df.shape*

Note that .shape has no parentheses and is a simple tuple of format (rows, columns). So we have 1000 rows and 11 columns in our movies DataFrame.

You'll be going to .shape a lot when cleaning and transforming data. For example, you might filter some rows based on some criteria and then want to know quickly how many rows were removed.

* **Handling duplicates**

This dataset does not have duplicate rows, but it is always important to verify you aren't aggregating duplicate rows.

To demonstrate, let's simply just double up our movies DataFrame by appending it to itself:

*temp\_df = movies\_df.append(movies\_df) temp\_df.shape*

*Out:*

*(2000, 11)*

Using append() will return a copy without affecting the original DataFrame. We are capturing this copy in temp so we aren't working with the real data.

Notice call .shape quickly proves our DataFrame rows have doubled. Now we can try dropping duplicates:

*temp\_df = temp\_df.drop\_duplicates()*

*temp\_df.shape*

*Out:*

*(1000, 11)*

Just like append(), the drop\_duplicates() method will also return a copy of your DataFrame, but this time with duplicates removed. Calling .shape confirms we're back to the 1000 rows of our original dataset.

It's a little verbose to keep assigning DataFrames to the same variable like in this example. For this reason, pandas has the inplace keyword argument on many of its methods. Using inplace=True will modify the DataFrame object in place:

*temp\_df.drop\_duplicates(inplace=True)*

Now our temp\_df *will* have the transformed data automatically.

Another important argument for drop\_duplicates() is keep, which has three possible options:

* + first: (default) Drop duplicates except for the first occurrence.
  + last: Drop duplicates except for the last occurrence.
  + False: Drop all duplicates.

Since we didn't define the keep arugment in the previous example it was defaulted to first. This means that if two rows are the same pandas will drop the second row and keep the first row. Using last has the opposite effect: the first row is dropped.

keep, on the other hand, will drop all duplicates. If two rows are the same then both will be dropped. Watch what happens to temp\_df:

temp\_df = movies\_df.append(movies\_df) # make a new copy temp\_df.drop\_duplicates(inplace=True, keep=False)

temp\_df.shape

Out:

(0, 11)

Since all rows were duplicates, keep=False dropped them all resulting in zero rows being left over. If you're wondering why you would want to do this, one reason is that it allows you to locate all duplicates in your dataset. When conditional selections are shown below you'll see how to do that.

### Column cleanup

Many times datasets will have verbose column names with symbols, upper and lowercase words, spaces, and typos. To make selecting data by column name easier we can spend a little time cleaning up their names.

Here's how to print the column names of our dataset:

movies\_df.columns

Out:

Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year', 'Runtime (Minutes)', 'Rating', 'Votes', 'Revenue (Millions)', 'Metascore'],

dtype='object')

Not only does .columns come in handy if you want to rename columns by allowing for simple copy and paste, it's also useful if you need to understand why you are receiving a Key Error when selecting data by column.

We can use the .rename() method to rename certain or all columns via a dict. We don't want parentheses, so let's rename those:

movies\_df.rename(columns={

'Runtime (Minutes)': 'Runtime',

'Revenue (Millions)': 'Revenue\_millions'

}, inplace=True)

movies\_df.columns

Out:

Index(['Rank', 'Genre', 'Description', 'Director', 'Actors', 'Year', 'Runtime',

'Rating', 'Votes', 'Revenue\_millions', 'Metascore'], dtype='object')

Excellent. But what if we want to lowercase all names? Instead of using .rename() we could also set a list of names to the columns like so:

movies\_df.columns = ['rank', 'genre', 'description', 'director', 'actors', 'year', 'runtime',

'rating', 'votes', 'revenue\_millions', 'metascore']

movies\_df.columns

Out:

Index(['rank', 'genre', 'description', 'director', 'actors', 'year', 'runtime',

'rating', 'votes', 'revenue\_millions', 'metascore'], dtype='object')

But that's too much work. Instead of just renaming each column manually we can do a list comprehension:

movies\_df.columns = [col.lower() for col in movies\_df] movies\_df.columns

Out:

Index(['rank', 'genre', 'description', 'director', 'actors', 'year', 'runtime',

'rating', 'votes', 'revenue\_millions', 'metascore'], dtype='object')

list (and dict) comprehensions come in handy a lot when working with pandas and data in general.

It's a good idea to lowercase, remove special characters, and replace spaces with underscores if you'll be working with a dataset for some time.

* **How to work with missing values**

When exploring data, you‘ll most likely encounter missing or null values, which are essentially placeholders for non-existent values. Most commonly you'll see Python's None or NumPy's np.nan, each of which are handled differently in some situations.

There are two options in dealing with nulls:

* 1. Get rid of rows or columns with nulls
  2. Replace nulls with non-null values, a technique known as imputation

Let's calculate to total number of nulls in each column of our dataset. The first step is to check which cells in our DataFrame are null:

movies\_df.isnull()

Notice isnull() returns a DataFrame where each cell is either True or False depending on that cell's null status.

To count the number of nulls in each column we use an aggregate function for summing: movies\_df.isnull().sum()

* **DataFrame slicing, selecting, extracting**

Up until now we've focused on some basic summaries of our data. We've learned about simple column extraction using single brackets, and we imputed null values in a column using fillna(). Below are the other methods of slicing, selecting, and extracting you'll need to use constantly.

It's important to note that, although many methods are the same, DataFrames and Series have different attributes, so you'll need be sure to know which type you are working with or else you will receive attribute errors.

Let's look at working with columns first. By column

You already saw how to extract a column using square brackets like this: genre\_col = movies\_df['genre']

type(genre\_col)

***Importance of Matplotlib library**

To make necessary statistical inferences, it becomes necessary to visualize your data and Matplotlib is one such solution for the Python users. It is a very powerful plotting library useful for those working with Python and NumPy. The most used module of Matplotib is Pyplot which provides an interface like MATLAB but instead, it uses Python and it is open source.

* **General Concepts**

A Matplotlib figure can be categorized into several parts as below:

1. Figure: It is a whole figure which may contain one or more than one axes (plots). You can think of a Figure as a canvas which contains plots.
2. Axes: It is what we generally think of as a plot. A Figure can contain many Axes. It contains two or three (in the case of 3D) Axis objects. Each Axes has a title, an x-label and a y-label.
3. Axis: They are the number line like objects and take care of generating the graph limits.
4. Artist: Everything which one can see on the figure is an artist like Text objects, Line2D objects, collection objects. Most Artists are tied to Axes.

***Matplotlib Library**

Pyplot is a module of Matplotlib which provides simple functions to add plot elements like lines, images, text, etc. to the current axes in the current figure.

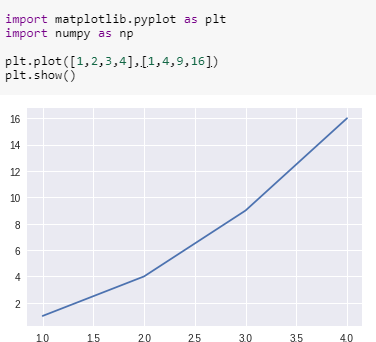
* **Make a simple plot**

import matplotlib.pyplot as plt import numpy as np

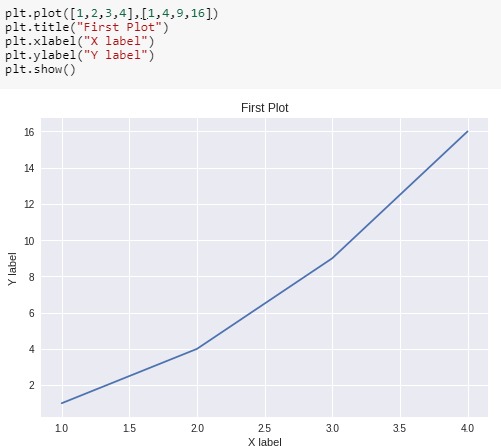
List of all the methods as they appeared.

* + plot(x-axis values, y-axis values) — plots a simple line graph with x-axis values against y-axis values
  + show() — displays the graph
  + title(―string‖) — set the title of the plot as specified by the string
  + xlabel(―string‖) — set the label for x-axis as specified by the string
  + ylabel(―string‖) — set the label for y-axis as specified by the string
  + figure() — used to control a figure level attributes
  + subplot(nrows, ncols, index) — Add a subplot to the current figure
  + suptitle(―string‖) — It adds a common title to the figure specified by the string
  + subplots(nrows, ncols, figsize) — a convenient way to create subplots, in a single call. It returns a tuple of a figure and number of axes.
  + set\_title(―string‖) — an axes level method used to set the title of subplots in a figure
  + bar(categorical variables, values, color) — used to create vertical bar graphs
  + barh(categorical variables, values, color) — used to create horizontal bar graphs
  + legend(loc) — used to make legend of the graph
  + xticks(index, categorical variables) — Get or set the current tick locations and labels of the x-axis
  + pie(value, categorical variables) — used to create a pie chart
  + hist(values, number of bins) — used to create a histogram
  + xlim(start value, end value) — used to set the limit of values of the x-axis
  + ylim(start value, end value) — used to set the limit of values of the y-axis
  + scatter(x-axis values, y-axis values) — plots a scatter plot with x-axis values against y-axis values
  + axes() — adds an axes to the current figure
  + set\_xlabel(―string‖) — axes level method used to set the x-label of the plot specified as a string
  + set\_ylabel(―string‖) — axes level method used to set the y-label of the plot specified as a string
  + scatter3D(x-axis values, y-axis values) — plots a three-dimensional scatter plot with x-axis values against y-axis values
  + plot3D(x-axis values, y-axis values) — plots a three-dimensional line graph with x- axis values against y-axis values

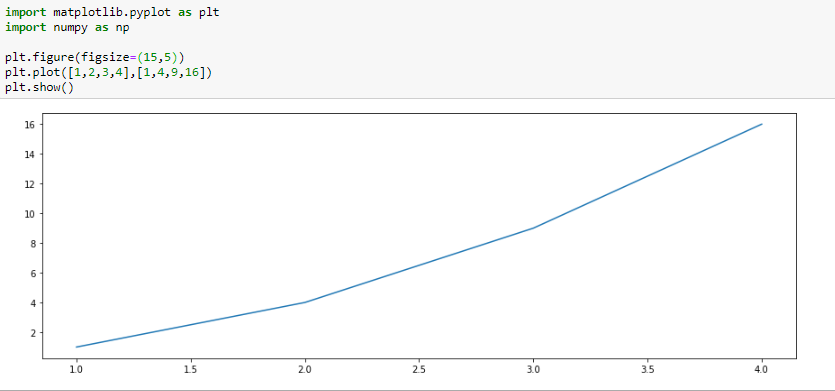
Here we import Matplotlib‘s Pyplot module and Numpy library as most of the data that we will be working with will be in the form of arrays only.



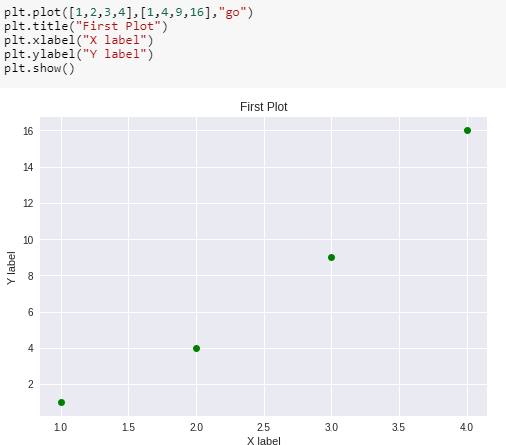
We pass two arrays as our input arguments to Pyplot‘s plot() method and use show() method to invoke the required plot. Here note that the first array appears on the x-axis and second array appears on the y-axis of the plot. Now that our first plot is ready, let us add the title, and name x-axis and y-axis using methods title(), xlabel() and ylabel() respectively.



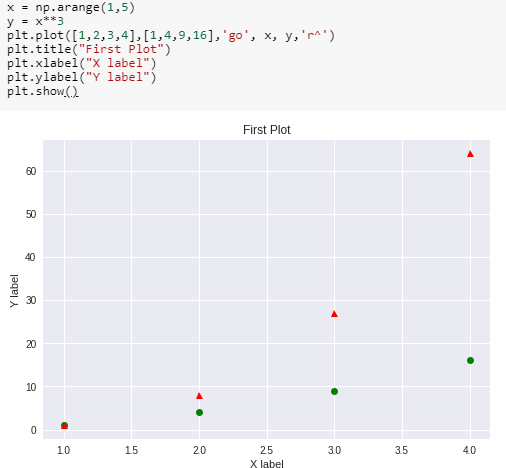
We can also specify the size of the figure using method figure() and passing the values as a tuple of the length of rows and columns to the argument figsize



With every X and Y argument, you can also pass an optional third argument in the form of a string which indicates the colour and line type of the plot. The default format is **b-** which means a solid blue line. In the figure below we use **go** which means green circles. Likewise, we can make many such combinations to format our plot.

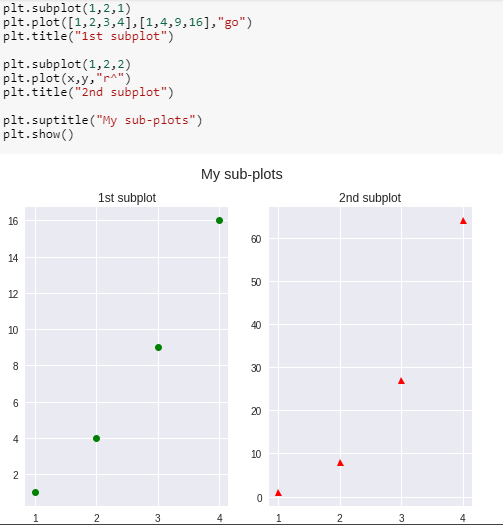


We can also plot multiple sets of data by passing in multiple sets of arguments of X and Y axis in the plot() method as shown.



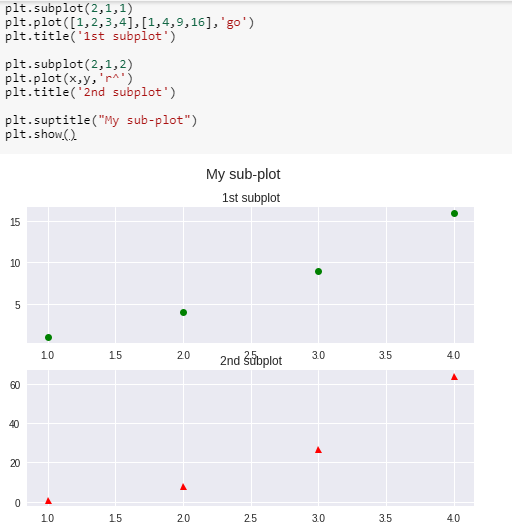
* **Multiple plots in one figure:**

We can use subplot() method to add more than one plots in one figure. In the image below, we used this method to separate two graphs which we plotted on the same axes in the previous example. The subplot() method takes three arguments: they are nrows, ncols and index. They indicate the number of rows, number of columns and the index number of the sub-plot. For instance, in our example, we want to create two sub-plots in one figure such that it comes in one row and in two columns and hence we pass arguments (1,2,1) and (1,2,2) in the subplot() method. Note that we have separately used title()method for both the subplots. We use suptitle() method to make a centralized title for the figure.

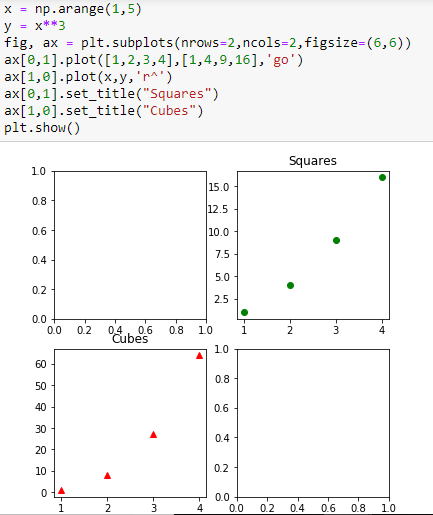


If we want our sub-plots in two rows and single column, we can pass arguments (2,1,1)

and (2,1,2)



The above way of creating subplots becomes a bit tedious when we want many subplots in our figure. A more convenient way is to use subpltots() method. Notice the difference of **‘s’** in both the methods. This method takes two arguments nrows and ncols as number of rows and number of columns respectively. This method creates two objects:figure and axes which we store in variables fig and ax which can be used to change the figure and axes level attributes respectively. Note that these variable names are chosen arbitrarily.



## Experiment No: 4

**Study of Python Libraries for Multiple Linear Regression**

***Machine Learning Library Scikit-Learn**

Scikit-learn was initially developed by David Cournapeau as a Google summer of code project in 2007. Later Matthieu Brucher joined the project and started to use it as apart of his thesis work. In 2010 INRIA got involved and the first public release (v0.1 beta) was published in late January 2010.The project now has more than 30 active contributors and has had paid sponsorship from INRIA, Google, Tinyclues and the Python Software Foundation.

Scikit-learn provides a range of supervised and unsupervised learning algorithms via a consistent interface in Python.

It is licensed under a permissive simplified BSD license and is distributed under many Linux distributions, encouraging academic and commercial use.

The library is built upon the SciPy (Scientific Python) that must be installed before you can use scikit-learn. This stack that includes:

* + - **NumPy**: Base n-dimensional array package
    - **SciPy**: Fundamental library for scientific computing
    - **Matplotlib**: Comprehensive 2D/3D plotting
    - **IPython**: Enhanced interactive console
    - **Sympy**: Symbolic mathematics
    - **Pandas**: Data structures and analysis

Extensions or modules for SciPy care conventionally named [SciKits.](http://scikits.appspot.com/scikits) As such, the module provides learning algorithms and is named scikit-learn.

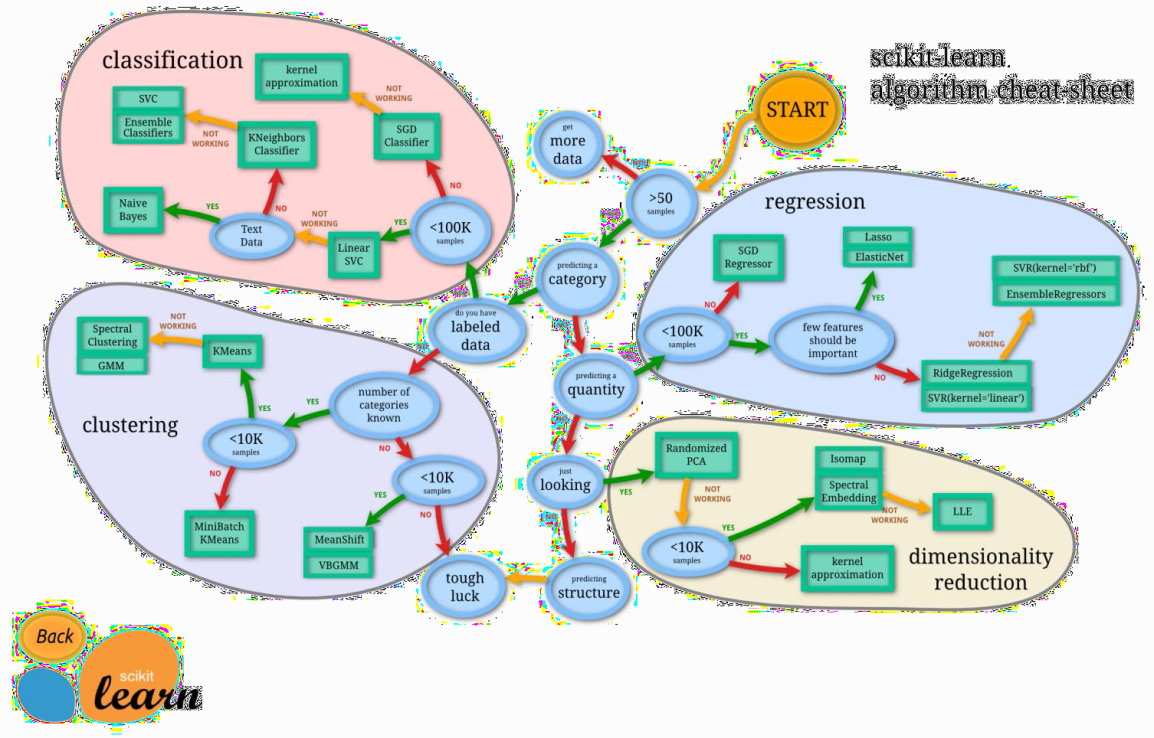
The vision for the library is a level of robustness and support required for use in production systems. This means a deep focus on concerns such as easy of use, code quality, collaboration, documentation and performance.

The library is focused on modeling data. It is not focused on loading, manipulating and summarizing data. For these features, refer to NumPy and Pandas.

***Enlist ML algorithms in sklearn library**

Some popular groups of models provided by scikit-learn include:

* + **Clustering**: for grouping unlabeled data such as KMeans.
  + **Cross Validation**: for estimating the performance of supervised models on unseen data.
  + **Datasets**: for test datasets and for generating datasets with specific properties for investigating model behavior. Eg. Boston Housing price, Iris Dataset
  + **Dimensionality Reduction**: for reducing the number of attributes in data for summarization, visualization and feature selection such as Principal component analysis.
  + **Ensemble methods**: for combining the predictions of multiple supervised models.
  + **Feature extraction**: for defining attributes in image and text data.
  + **Feature selection**: for identifying meaningful attributes from which to create supervised models.
  + **Parameter Tuning**: for getting the most out of supervised models.
  + **Manifold Learning**: For summarizing and depicting complex multi-dimensional data.
  + **Supervised Models**: a vast array not limited to generalized linear models, discriminate analysis, naive bayes, lazy methods, neural networks, support vector machines and decision trees.



***Multiple Regression Analysis**

The multiple linear regression explains the relationship between one continuous dependent variable (y) and two or more independent variables (x1, x2, x3… etc). Note that it says CONTINUOUS dependant variable. Since y is the sum of beta, beta1 x1, beta2 x2 etc etc, the resulting y will be a number, a continuous variable, instead of a ―yes‖, ―no‖ answer (categorical).

Multiple linear regression (MLR), also known simply as multiple regression, is a statistical technique that uses several explanatory variables to predict the outcome of a response variable. The goal of multiple linear regression (MLR) is to model the linear relationship between the explanatory (independent) variables and response (dependent) variable.

In essence, multiple regression is the extension of ordinary least-squares (OLS) regression that involves more than one explanatory variable.

The multiple regression model is based on the following assumptions:

* + - There is a linear relationship between the dependent variables and the independent variables.
    - The independent variables are not too highly correlated with each other. (Absence of multicollinearity)
    - yi observations are selected independently and randomly from the population.
    - Residuals should be normally distributed with a mean of 0 and variance *σ.*

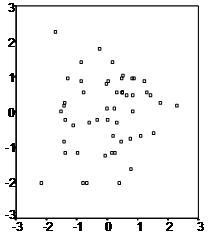
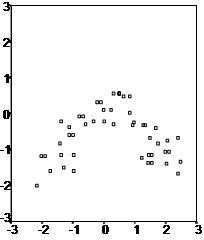
Multiple Linear Regression attempts to model the Relationship between two or more features and a response by fitting a linear equation to observed data. The steps to perform multiple linear Regression are almost similar to that of simple linear Regression. The Difference Lies in the Evalution. We can use it to find out which factor has the highest impact on the predicted output and now different variable relate to each other.

Here : **Y= b0 + b1\*x1 + b2\*x2 + b3\*x3 +…… bn\*xn**

Y = Dependent variable and x1, x2, x3, …… xn = multiple independent variables

### First assumption

Linear regression needs the relationship between the independent and dependent variables to be linear. It is also important to check for outliers since linear regression is sensitive to outlier effects. The linearity assumption can best be tested with scatter plots, the following two examples depict two cases, where no and little linearity is present.



***Multicollinearity and Correlation measure**

In regression, multicollinearity or collinearity refers to the extent to which independent variables are correlated. Multicollinearity exists when:

* + One independent variable is correlated with another independent variable.
  + One independent variable is correlated with a linear combination of two or more independent variables.

Multicollinearity may be tested with three central criteria:

1. Correlation matrix – when computing the matrix of Pearson‘s Bivariate Correlation among all independent variables the correlation coefficients need to be smaller than 1.
2. Tolerance – the tolerance measures the influence of one independent variable on all other independent variables; the tolerance is calculated with an initial linear regression analysis. Tolerance is defined as T = 1 – R² for these first step regression analysis. With T < 0.1 there might be multicollinearity in the data and with T < 0.01 there certainly is.
3. Variance Inflation Factor (VIF) – the variance inflation factor of the linear regression is defined as VIF = 1/T. With VIF > 5 there is an indication that multicollinearity may be present; with VIF

> 10 there is certainly multicollinearity among the variables.

Checking for collinearity helps you get rid of variables that are skewing your data by having a significant relationship with another variable.

Correlation between variables describe the relationship between two variables. If they are extremely correlated, then they are collinear.

Autocorrelation occurs when a variable‘s data affects another instance of that same variable (same column, different row). Linear regression only works if there is little or no autocorrelation in the dataset, and each instance is independent of each other. If instances are autocorrelated then your residuals are not independent from each other, and will show a pattern.

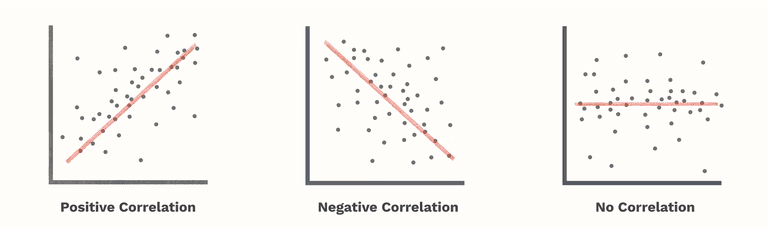
Multicollinearity exists when two or more of the predictors (x variables) in a regression model are moderately or highly correlated (different column). When one of our predictors is able to strongly predict another predictor or have weird relationships with each other (maybe x2 = x3 or x2 = 2(x3) + x4), then your regression equation is going to be a mess.

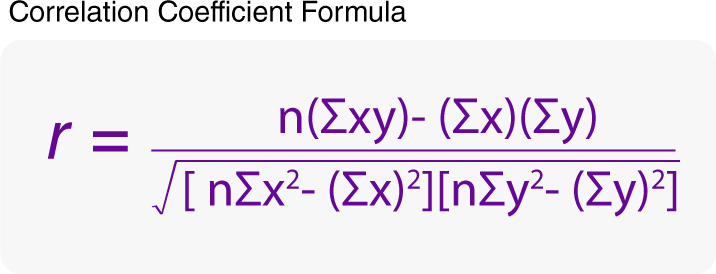
The simplest method to detect collinearity would be to plot it out in graphs or to view a correlation matrix to check out pairwise correlation (correlation between 2 variables). If you have two variables that are highly correlated, your best course of action is to just remove one of them.

Why is multicollinearity an issue with regression? Well, the regression equation is the best fit line to represent the effects of your predictors and the dependant variable, and does not include the effects of one predictor on another.

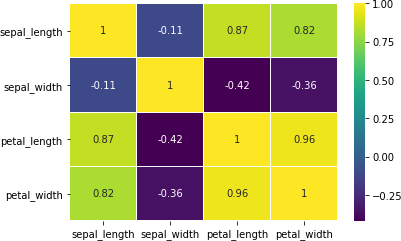
Having high collinearity (correlation of 1.00) between predictors will affect your coefficients and the accuracy, plus its ability to reduce the SSE (sum of squared errors — that thing you need to minimise with your regression).

***Correlation Coefficient and Correlation Matrix**

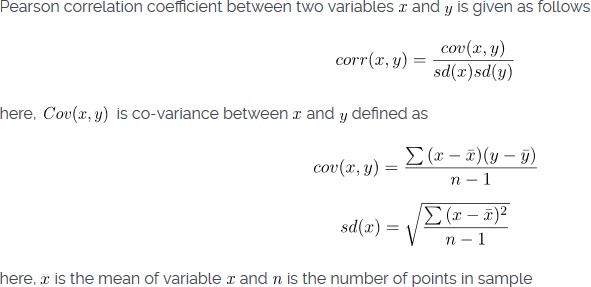
The correlation coefficient, denoted by r tells us how closely data in a scatterplot fall along a straight line. The closer that the absolute value of r is to one, the better that the data are described by a linear equation. If r =1 or r = -1 then the data set is perfectly aligned. Data sets with values of r close to zero show little to no straight-line relationship.



A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. A correlation matrix is used as a way to summarize data.



In Statistics, Pearson correlation coefficient is widely used to find out relationship among random variables. In the above example a correlation matrix is represented using heat matrix (map).



***The coefficient of determination (R-squared)**

The coefficient of determination (R-squared) is a statistical metric that is used to measure how much of the variation in outcome can be explained by the variation in the independent variables. R2 always increases as more predictors are added to the MLR model even though the predictors may not be related to the outcome variable.

R2 by itself can't thus be used to identify which predictors should be included in a model and which should be excluded. R2 can only be between 0 and 1, where 0 indicates that the outcome cannot be predicted by any of the independent variables and 1 indicates that the outcome can be predicted without error from the independent variables.

When interpreting the results of a multiple regression, beta coefficients are valid while holding all other variables constant ("all else equal"). The output from a multiple regression can be displayed horizontally as an equation, or vertically in table form.

**Experiment No: 5**

Title: DESCRIPTION:

DESCRIPTION:

**Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.**

import numpy as np import math

from data\_loader import read\_data

class Node:

def init (self, attribute): self.attribute = attribute self.children = [] self.answer = ""

def str (self): return self.attribute

def subtables(data, col, delete): dict = {}

items = np.unique(data[:, col])

count = np.zeros((items.shape[0], 1), dtype=np.int32) for x in range(items.shape[0]):

for y in range(data.shape[0]):

if data[y, col] == items[x]: count[x] += 1

for x in range(items.shape[0]):

dict[items[x]] = np.empty((int(count[x]), data.shape[1]), dtype="|S32")

pos = 0

for y in range(data.shape[0]): if data[y, col] == items[x]:

dict[items[x]][pos] = data[y] pos += 1

if delete:

dict[items[x]] = np.delete(dict[items[x]], col, 1) return items, dict

def entropy(S):

items = np.unique(S) if items.size == 1:

return 0

counts = np.zeros((items.shape[0], 1)) sums = 0

for x in range(items.shape[0]):

counts[x] = sum(S == items[x]) / (S.size \* 1.0)

for count in counts:

sums += -1 \* count \* math.log(count, 2) return sums

def gain\_ratio(data, col):

items, dict = subtables(data, col, delete=False)

total\_size = data.shape[0]

entropies = np.zeros((items.shape[0], 1)) intrinsic = np.zeros((items.shape[0], 1)) for x in range(items.shape[0]):

ratio = dict[items[x]].shape[0]/(total\_size \* 1.0) entropies[x] = ratio \* entropy(dict[items[x]][:, -1]) intrinsic[x] = ratio \* math.log(ratio, 2)

total\_entropy = entropy(data[:, -1]) iv = -1 \* sum(intrinsic)

for x in range(entropies.shape[0]): total\_entropy -= entropies[x]

return total\_entropy / iv

def create\_node(data, metadata):

if (np.unique(data[:, -1])).shape[0] == 1: node = Node("")

node.answer = np.unique(data[:, -1])[0] return node

gains = np.zeros((data.shape[1] - 1, 1)) for col in range(data.shape[1] - 1):

gains[col] = gain\_ratio(data, col) split = np.argmax(gains)

node = Node(metadata[split])

metadata = np.delete(metadata, split, 0)

items, dict = subtables(data, split, delete=True)

for x in range(items.shape[0]):

child = create\_node(dict[items[x]], metadata) node.children.append((items[x], child))

return node def empty(size):

s = ""

for x in range(size): s += " "

return s

def print\_tree(node, level): if node.answer != "":

print(empty(level), node.answer) return

print(empty(level), node.attribute) for value, n in node.children:

print(empty(level + 1), value) print\_tree(n, level + 2)

metadata, traindata = read\_data("tennis.csv") data = np.array(traindata)

node = create\_node(data, metadata) print\_tree(node, 0)

import csv

def read\_data(filename):

with open(filename, 'r') as csvfile:

datareader = csv.reader(csvfile, delimiter=',') headers = next(datareader)

metadata = [] traindata = []

for name in headers: metadata.append(name)

for row in datareader: traindata.append(row)

return (metadata, traindata)

## Tennis.csv

outlook,temperature,humidity,wind, answer sunny,hot,high,weak,no sunny,hot,high,strong,no overcast,hot,high,weak,yes rain,mild,high,weak,yes rain,cool,normal,weak,yes rain,cool,normal,strong,no overcast,cool,normal,strong,yes sunny,mild,high,weak,no sunny,cool,normal,weak,yes rain,mild,normal,weak,yes sunny,mild,normal,strong,yes overcast,mild,high,strong,yes overcast,hot,normal,weak,yes rain,mild,high,strong,no

## Output

outlook

overcast b'yes'

rain

wind

b'strong' b'no' b'weak' b'yes'

sunny

humidity b'high' b'no'

b'normal' b'yes

## Experiment No: 6

## Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets.

import csv import random import math

def loadCsv(filename):

lines = csv.reader(open(filename, "r")); dataset = list(lines)

for i in range(len(dataset)):

#converting strings into numbers for processing dataset[i] = [float(x) for x in dataset[i]]

return dataset

def splitDataset(dataset, splitRatio): #67% training size

trainSize = int(len(dataset) \* splitRatio); trainSet = []

copy = list(dataset);

while len(trainSet) < trainSize:

#generate indices for the dataset list randomly to pick ele for training data index = random.randrange(len(copy)); trainSet.append(copy.pop(index))

return [trainSet, copy]

def separateByClass(dataset):

separated = {}

#creates a dictionary of classes 1 and 0 where the values are the instacnes belonging to each class

for i in range(len(dataset)): vector = dataset[i]

if (vector[-1] not in separated): separated[vector[-1]] = []

separated[vector[-1]].append(vector) return separated

def mean(numbers):

return sum(numbers)/float(len(numbers))

def stdev(numbers):

avg = mean(numbers)

variance = sum([pow(x-avg,2) for x in numbers])/float(len(numbers)-1) return math.sqrt(variance)

def summarize(dataset):

summaries = [(mean(attribute), stdev(attribute)) for attribute in zip(\*dataset)]; del summaries[-1]

return summaries

def summarizeByClass(dataset):

separated = separateByClass(dataset); summaries = {}

for classValue, instances in separated.items():

#summaries is a dic of tuples(mean,std) for each class value summaries[classValue] = summarize(instances)

return summaries

def calculateProbability(x, mean, stdev):

exponent = math.exp(-(math.pow(x-mean,2)/(2\*math.pow(stdev,2)))) return (1 / (math.sqrt(2\*math.pi) \* stdev)) \* exponent

def calculateClassProbabilities(summaries, inputVector):

probabilities = {}

for classValue, classSummaries in summaries.items():#class and attribute information as mean and sd

probabilities[classValue] = 1

for i in range(len(classSummaries)):

mean, stdev = classSummaries[i] #take mean and sd of every attribute for class 0 and 1 seperaely

x = inputVector[i] #testvector's first attribute probabilities[classValue] \*= calculateProbability(x, mean, stdev);#use

normal dist

return probabilities

def predict(summaries, inputVector):

probabilities = calculateClassProbabilities(summaries, inputVector) bestLabel, bestProb = None, -1

for classValue, probability in probabilities.items():#assigns that class which has the

highest prob

if bestLabel is None or probability > bestProb: bestProb = probability

bestLabel = classValue return bestLabel

def getPredictions(summaries, testSet): predictions = []

for i in range(len(testSet)):

result = predict(summaries, testSet[i]) predictions.append(result)

return predictions

def getAccuracy(testSet, predictions):

correct = 0

for i in range(len(testSet)):

if testSet[i][-1] == predictions[i]: correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

filename = '5data.csv' splitRatio = 0.67

dataset = loadCsv(filename);

trainingSet, testSet = splitDataset(dataset, splitRatio)

print('Split {0} rows into train={1} and test={2} rows'.format(len(dataset), len(trainingSet), len(testSet)))

# prepare model

summaries = summarizeByClass(trainingSet); # test model

predictions = getPredictions(summaries, testSet) accuracy = getAccuracy(testSet, predictions)

print('Accuracy of the classifier is : {0}%'.format(accuracy)) main()

## Output

confusion matrix is as follows [[17 0 0]

[ 0 17 0]

[ 0 0 11]]

Accuracy metrics

precision recall f1-score support

|  |  |
| --- | --- |
| 0 1.00 1.00 1.00 | 17 |
| 1 1.00 1.00 1.00 | 17 |
| 2 1.00 1.00 1.00 | 11 |

avg / total 1.00 1.00 1.00 45

## Experiment No: 7

## Write a program to construct a Bayesian network considering medical data. Use this model to demonstrate the diagnosis of heart patients using standard Heart Disease Data Set. You can use Java/Python ML library classes/API.

From pomegranate import\* Asia=DiscreteDistribution({ „True‟:0.5, „False‟:0.5 }) Tuberculosis=ConditionalProbabilityTable(

[[ „True‟, „True‟, 0.2],

[„True‟, „False‟, 0.8],

[ „False‟, „True‟, 0.01],

[ „False‟, „False‟, 0.98]], [asia])

Smoking = DiscreteDistribution({ „True‟:0.5, „False‟:0.5 }) Lung = ConditionalProbabilityTable(

[[ „True‟, „True‟, 0.75],

[„True‟, „False‟,0.25].

[ „False‟, „True‟, 0.02],

[ „False‟, „False‟, 0.98]], [ smoking])

Bronchitis = ConditionalProbabilityTable( [[ „True‟, „True‟, 0.92],

[„True‟, „False‟,0.08].

[ „False‟, „True‟,0.03],

[ „False‟, „False‟, 0.98]], [ smoking])

Tuberculosis\_or\_cancer = ConditionalProbabilityTable( [[ „True‟, „True‟, „True‟, 1.0],

[„True‟, „True‟, „False‟, 0.0],

[„True‟, „False‟, „True‟, 1.0],

[„True‟, „False‟, „False‟, 0.0],

[„False‟, „True‟, „True‟, 1.0],

[„False‟, „True‟, „False‟, 0.0],

[„False‟, „False‟ „True‟, 1.0],

[„False‟, „False‟, „False‟, 0.0]], [tuberculosis, lung])

Xray = ConditionalProbabilityTable( [[ „True‟, „True‟, 0.885],

[„True‟, „False‟, 0.115],

[ „False‟, „True‟, 0.04],

[ „False‟, „False‟, 0.96]], [tuberculosis\_or\_cancer]) dyspnea = ConditionalProbabilityTable(

[[ „True‟, „True‟, „True‟, 0.96],

[„True‟, „True‟, „False‟, 0.04],

[„True‟, „False‟, „True‟, 0.89],

[„True‟, „False‟, „False‟, 0.11],

[„False‟, „True‟, „True‟, 0.96],

[„False‟, „True‟, „False‟, 0.04],

[„False‟, „False‟ „True‟, 0.89],

[„False‟, „False‟, „False‟, 0.11 ]], [tuberculosis\_or\_cancer, bronchitis]) s0 = State(asia, name=”asia”)

s1 = State(tuberculosis, name=” tuberculosis”) s2 = State(smoking, name=” smoker”)

network = BayesianNetwork(“asia”) network.add\_nodes(s0,s1,s2) network.add\_edge(s0,s1) network.add\_edge(s1.s2) network.bake()

print(network.predict\_probal({„tuberculosis‟: „True‟}))

## Experiment No: 8

## Write a program to implement *k*-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Java/Python ML library classes can be used for this problem.

import csv import random import math import operator

def loadDataset(filename, split, trainingSet=[] , testSet=[]): with open(filename, 'rb') as csvfile:

lines = csv.reader(csvfile) dataset = list(lines)

for x in range(len(dataset)-1): for y in range(4):

dataset[x][y] = float(dataset[x][y]) if random.random() < split:

trainingSet.append(dataset[x]) else:

testSet.append(dataset[x])

def euclideanDistance(instance1, instance2, length): distance = 0

for x in range(length):

distance += pow((instance1[x] - instance2[x]), 2) return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k): distances = []

length = len(testInstance)-1

for x in range(len(trainingSet)):

dist = euclideanDistance(testInstance, trainingSet[x], length) distances.append((trainingSet[x], dist))

distances.sort(key=operator.itemgetter(1)) neighbors = []

for x in range(k):

neighbors.append(distances[x][0]) return neighbors

def getResponse(neighbors): classVotes = {}

for x in range(len(neighbors)): response = neighbors[x][-1] if response in classVotes:

classVotes[response] += 1

else:

classVotes[response] = 1

sortedVotes =

sorted(classVotes.iteritems(),

reverse=True)

return sortedVotes[0][0]

def getAccuracy(testSet, predictions): correct = 0 for x in range(len(testSet)): key=operator.itemgetter(1

),

if testSet[x][-1] == predictions[x]: correct += 1

return (correct/float(len(testSet))) \* 100.0

def main():

# prepare data trainingSet= [] testSet=[] split = 0.67

loadDataset('knndat.data', split, trainingSet, testSet) print('Train set: ' + repr(len(trainingSet))) print('Test set: ' + repr(len(testSet)))

# generate predictions predictions=[] k=3

for x in range(len(testSet)):

neighbors = getNeighbors(trainingSet, testSet[x], k) result = getResponse(neighbors) predictions.append(result)

print('> predicted=' + repr(result) + ', actual=' + repr(testSet[x][- 1])) accuracy = getAccuracy(testSet, predictions)

print('Accuracy: ' + repr(accuracy) + '%') main()

## OUTPUT

**Confusion matrix is as follows**

[[11 0 0]

[0 9 1]

[0 1 8]]

Accuracy metrics

0 1.00 1.00 1.00 11

1 0.90 0.90 0.90 10

2 0.89 0.89 0,89 9

Avg/Total 0.93 0.93 0.93 30