Python Data Science

Introduction

- NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays.
- Using NumPy, mathematical and logical operations on arrays can be performed.
- It is an extension module for Python, mostly written in C.
- This makes sure that the precompiled mathematical and numerical functions and functionalities of Numpy guarantee great execution speed.
- Furthermore, NumPy enriches the programming language Python with powerful data structures, implementing multi-dimensional arrays and matrices.
- These data structures guarantee efficient calculations with matrices and arrays.
- The implementation is even aiming at huge matrices and arrays, better known under the heading of "big data".
- Besides that, the module supplies a large library of high-level mathematical functions to operate on these matrices and arrays.

Why NumPy?

- It internally stores data in contiguous block of memory, independent of other built-in Python Objects.
- NumPy operations perform complex computations on entire arrays without the need for Python for loops.

Installation

- Install numpy and scipy.
- In windows,
  - Click Command prompt
  - Go to the folder that has pip
  - Type pip install numpy
  - Type pip install scipy
- In ubuntu
  - At the terminal type
    - sudo apt-get install numpy
    - sudo apt-get install scipy

Numpy Arrays

- At the core, numpy provides the excellent ndarray objects, short for n-dimensional arrays.
- In a ‘ndarray’ object, aka ‘array’, multiple items of the same data type can be stored.

ϖ It is the facilities around the array object that makes numpy so convenient for performing math and data manipulations.

- δ **numpy.array(object, dtype = None, copy = True, order = None, subok = False, ndmin = 0)**
  - ∇ object
    - ₪ Any object exposing the array interface method returns an array, or any (nested) sequence.
  - ∇ dtype
    - ₪ Desired data type of array, optional
  - ∇ copy
    - ₪ Optional. By default (true), the object is copied
  - ∇ order
    - ₪ C (row major) or F (column major) or A (any) (default)
  - ∇ subok
    - ₪ By default, returned array forced to be a base class array. If true, sub-classes passed through
  - ∇ ndmin
    - ₪ Specifies minimum dimensions of resultant array

Data Types

Type	Type Code	Description
int8, uint8	i1, u1	Signed and unsigned 8-bit integer types
int16, uint16	i2, u2	Signed and unsigned 16-bit integer types
int32, uint32	i4, u4	Signed and unsigned 32-bit integer types
int64, uint64	i8, u8	Signed and unsigned 64-bit integer types
float16	f2	Half-precision floating point
float32	f4 or f	Standard single-precision floating point
float64	f8 or d	Standard double-precision floating point
float128	f16 or g	Extended precision floating point
complex64, complex128, complex256	c8, c16, c32	Complex numbers
Bool	?	Boolean Type
Object	0	A value can be any Python object
string_	S	Fixed-length ASCII string type (Ex – S10)

unicode_	U	Fixed-length Unicode type (Ex – U10)

Other methods to create arrays

Function	Description
array	Convert input data to an ndarray either by implicitly identifying the dtype or or explicitly specifying the dtype. Copies the input data by default.
asanyarray	Convert input to ndarray Do not copy if the input is already an ndarray.
arange	Returns a ndarray
ones, ones_like	Produce an array of all 1's of given shape and dtype. ones_like takes another array and produces a ones array of the same shape and dtype
zeroes, zeroes_like	Produce an array of all 0's of given shape and dtype. zeroes_like takes another array and produces a zeroes array of the same shape and dtype
empty, empty_like	Creates new array by allocating new memory but does not populate with values
full, full_like	Produces an array of given shape and dtype with all values set to the indicated fill value
eye, identity	Create a square n * n identity matrix

Arrays over Lists

- Arrays support vectorised operations, while lists don't.
- Once an array is created, you cannot change its size. You will have to create a new array or overwrite the existing one.
- Every array has one and only one dtype. All items in it should be of that dtype.
- An equivalent numpy array occupies much less space than a python list of lists.

Information from an array

- If a numpy vector that was not created by yourself is given.
- What are the things needed to explore in order to know about that array?

- δ If it is a 1D or a 2D array or more.
- δ How many items are present in each dimension?
- δ What is its datatype?
- δ What is the total number of items in it?
- δ Samples of first few items in the array?

- ϖ Shape
  - δ <array_name>.shape
- ϖ Data type
  - δ <array_name>.dtype
- ϖ Size
  - δ <array_name>.size
- ϖ Number of Dimensions
  - δ <array_name>.ndim

Extract Specific Information

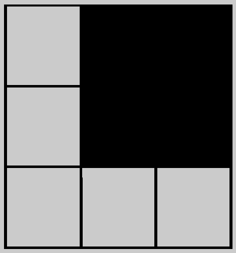
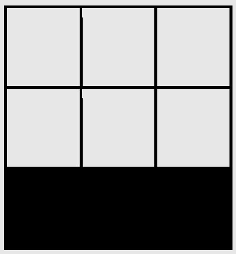
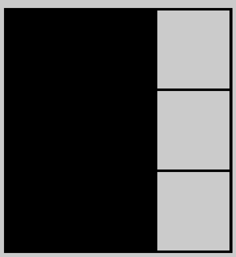
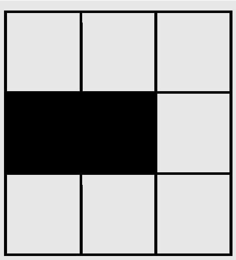
- ϖ Specific portions on an array using indexing starting with 0 can be extracted.
- ϖ But unlike lists, numpy arrays can optionally accept as many parameters in the square brackets as there is number of dimensions.

Basic Indexing and Slicing - One dimensional Array

- ϖ array_name[index]
- ϖ array_name[starting_pos, ending_pos]
  - δ It displays from starting_pos till ending_pos excluding ending_pos
- ϖ array_array_name[:]
  - δ Called bare slice. All elements.

Basic Indexing and Slicing - Two-dimensional Array

- ϖ array_name[row_index][col_index]
- ϖ array_name[row_index,col_index]
- ϖ array_name[slice_num: slice_num]
- ϖ array_name[:2]
  - δ Select first two rows of array_name
- ϖ array_name[1,:2]
  - δ Select second row but only first two columns
- ϖ array_name[:2,2]
  - δ Select third column but first two rows.

	Expression	Shape
	arr[:2, 1;]	(2,2)
	arr[2] arr[2,:] arr[2:, :]	(3,) (3,) (1, 3)
	arr[:, :2]	(3, 2)
	arr[1, :2] Arr[1:2, :2]	(2,) (1, 2)

Array Functions

ϖ Array_name.reshape(array_name, newshape, order)

- δ Array_name – input array
- δ Newshape - int or tuple of int. New shape should be compatible to the original shape
- δ Order - [C-contiguous, F-contiguous, A-contiguous; optional]
  - ∇ C-contiguous order in memory(last index varies the fastest)
  - ∇ C order means that operating row-rise on the array will be slightly quicker
  - ∇ FORTRAN-contiguous order in memory (first index varies the fastest).
  - ∇ F order means that column-wise operations will be faster.
  - ∇ 'A' means to read / write the elements in Fortran-like index order if array is Fortran contiguous in memory, C-like order otherwise

Basic Indexing and Slicing

- array_name[cond]
  - Use the contents based on a condition.
- array_name[cond, slice_num : slice_num]
  - Use specific columns with contents based on a condition
- array_name[!cond]
  - Use the contents that is not satisfying the condition.
- array_name[~(cond)]
  - Use the contents that is not satisfying the condition.
- Can use multiple conditions by using | and &.
- array_name[relational_condition]
- array_name[relational_condition] = new_value
- array_name[logical_condition] = new_value
  - Change the values of the indices that satisfy the condition

Fancy Indexing

- array_name[[row numbers,…..]]
  - Select out a subset of the rows in a particular order
- array_name[[-index number,………..]]
  - Select out a subset of the rows in a particular order given backwards
- array_name[[index number$_{i1}$, index number$_{i2}$ …, index number$_{in}$], [index number$_{j1}$, index number$_{j2}$ …, index number$_{jn}$]]
  - Elements (index number$_{i1}$, index number$_{j1}$), (index number$_{i2}$, index number$_{j2}$), ….. (index number$_{in}$, index number$_{jn}$) are selected
    - The result is a one-dimensional list

Matrix Operations

- <arrayname>.T
  - Gives the transpose of the matrix
- <array_name>.transpose(array_name, axes=None|list of integers to represent the axes)
  - Gives the transpose of the matrix with the dimensions of an array permuted.
- np.dot(array_name, array_name)
  - Matrix Product
- Array_name.swapaxes(axis$_1$, axis$_2$, …..)
  - Switch the indicated axes to rearrange the data

Universal Functions

- These functions perform element-wise operations on data in ndarrays and produce one or more scalar results.
- Also called as ufunc.
  - np.sqrt(array_name)
  - np.exp(array_name)

Mathematical functions

Function	Description
add(x1, x2, /[, out, where, casting, order, …])	Add arguments element-wise.
subtract(x1, x2, /[, out, where, casting, …])	Subtract arguments, element-wise.
multiply(x1, x2, /[, out, where, casting, …])	Multiply arguments element-wise.
divide(x1, x2, /[, out, where, casting, …])	Returns a true division of the inputs, element-wise.
logaddexp(x1, x2, /[, out, where, casting, …])	Logarithm of the sum of exponentiations of the inputs.
logaddexp2(x1, x2, /[, out, where, casting, …])	Logarithm of the sum of exponentiations of the inputs in base-2.
true_divide(x1, x2, /[, out, where, …])	Returns a true division of the inputs, element-wise.
floor_divide(x1, x2, /[, out, where, …])	Return the largest integer smaller or equal to the division of the inputs.
negative(x, /[, out, where, casting, order, …])	Numerical negative, element-wise.
positive(x, /[, out, where, casting, order, …])	Numerical positive, element-wise.
power(x1, x2, /[, out, where, casting, …])	First array elements raised to powers from second array, element-wise
remainder(x1, x2, /[, out, where, casting, …])	Return element-wise remainder of division.
mod(x1, x2, /[, out, where, casting, order, …])	Return element-wise remainder of division.
fmod(x1, x2, /[, out, where, casting, …])	Return the element-wise remainder of division.
divmod(x1, x2[, out1, out2], / [[, out, …])	Return element-wise quotient and remainder simultaneously.
absolute(x, /[, out, where, casting, order, …])	Calculate the absolute value element-wise.

fabs(x, /[, out, where, casting, order, …])	Compute the absolute values element-wise.
rint(x, /[, out, where, casting, order, …])	Round elements of the array to the nearest integer.
sign(x, /[, out, where, casting, order, …])	Returns an element-wise indication of the sign of a number.
heaviside(x1, x2, /[, out, where, casting, …])	Compute the Heaviside step function.
conj(x, /[, out, where, casting, order, …])	Return the complex conjugate, element-wise.
exp(x, /[, out, where, casting, order, …])	Calculate the exponential of all elements in the input array.
exp2(x, /[, out, where, casting, order, …])	Calculate 2**p for all p in the input array.
log(x, /[, out, where, casting, order, …])	Natural logarithm, element-wise.
log2(x, /[, out, where, casting, order, …])	Base-2 logarithm of x.
log10(x, /[, out, where, casting, order, …])	Return the base 10 logarithm of the input array, element-wise.
expm1(x, /[, out, where, casting, order, …])	Calculate exp(x) - 1 for all elements in the array.
log1p(x, /[, out, where, casting, order, …])	Return the natural logarithm of one plus the input array, element-wise.
sqrt(x, /[, out, where, casting, order, …])	Return the non-negative square-root of an array, element-wise.
square(x, /[, out, where, casting, order, …])	Return the element-wise square of the input.
cbrt(x, /[, out, where, casting, order, …])	Return the cube-root of an array, element-wise.
reciprocal(x, /[, out, where, casting, …])	Return the reciprocal of the argument, element-wise.
gcd(x1, x2, /[, out, where, casting, order, …])	Returns the greatest common divisor of |x1| and |x2|
lcm(x1, x2, /[, out, where, casting, order, …])	Returns the lowest common multiple of |x1| and |x2|

Trigonometric Functions

Function	Description
sin(x, /[, out, where, casting, order, …])	Trigonometric sine, element-wise.

cos(x, /[, out, where, casting, order, …])	Cosine element-wise.
tan(x, /[, out, where, casting, order, …])	Compute tangent element-wise.
arcsin(x, /[, out, where, casting, order, …])	Inverse sine, element-wise.
arccos(x, /[, out, where, casting, order, …])	Trigonometric inverse cosine, element-wise.
arctan(x, /[, out, where, casting, order, …])	Trigonometric inverse tangent, element-wise.
arctan2(x1, x2, /[, out, where, casting, …])	Element-wise arc tangent of x1/x2 choosing the quadrant correctly.
hypot(x1, x2, /[, out, where, casting, …])	Given the “legs” of a right triangle, return its hypotenuse.
sinh(x, /[, out, where, casting, order, …])	Hyperbolic sine, element-wise.
cosh(x, /[, out, where, casting, order, …])	Hyperbolic cosine, element-wise.
tanh(x, /[, out, where, casting, order, …])	Compute hyperbolic tangent element-wise.
arcsinh(x, /[, out, where, casting, order, …])	Inverse hyperbolic sine element-wise.
arccosh(x, /[, out, where, casting, order, …])	Inverse hyperbolic cosine, element-wise
arctanh(x, /[, out, where, casting, order, …])	Inverse hyperbolic tangent element-wise.
deg2rad(x, /[, out, where, casting, order, …])	Convert angles from degrees to radians.
rad2deg(x, /[, out, where, casting, order, …])	Convert angles from radians to degrees.

- All trigonometric functions use radians when an angle is called for.
- The ratio of degrees to radians is 180^{\circ}/\pi.

Bit-wise functions

- These function all require integer arguments and they manipulate the bit-pattern of those arguments

Functions	Description
bitwise_and(x1, x2, /[, out, where, …])	Compute the bit-wise AND of two arrays element-wise.
bitwise_or(x1, x2, /[, out, where, casting, …])	Compute the bit-wise OR of two arrays element-wise.
bitwise_xor(x1, x2, /[, out, where, …])	Compute the bit-wise XOR of two arrays element-wise.
invert(x, /[, out, where, casting, order, …])	Compute bit-wise inversion, or bit-wise NOT, element-wise.
left_shift(x1, x2, /[, out, where, casting, …])	Shift the bits of an integer to the left.
right_shift(x1, x2, /[, out, where, …])	Shift the bits of an integer to the right.

Comparison functions

Functions	Description
greater(x1, x2, /[, out, where, casting, …])	Return the truth value of (x1 > x2) element-wise
greater_equal(x1, x2, /[, out, where, …])	Return the truth value of (x1 >= x2) element-wise.
less(x1, x2, /[, out, where, casting, …])	Return the truth value of (x1 < x2) element-wise.
less_equal(x1, x2, /[, out, where, casting, …])	Return the truth value of (x1 =< x2) element-wise.
not_equal(x1, x2, /[, out, where, casting, …])	Return (x1 != x2) element-wise.
equal(x1, x2, /[, out, where, casting, …])	Return (x1 == x2) element-wise.
logical_and(x1, x2, /[, out, where, …])	Compute the truth value of x1 AND x2 element-wise.
logical_or(x1, x2, /[, out, where, casting, …])	Compute the truth value of x1 OR x2 element-wise.

logical_xor(x1, x2, /[, out, where, …])	Compute the truth value of x1 XOR x2, element-wise.
logical_not(x, /[, out, where, casting, …])	Compute the truth value of NOT x element-wise.

- Do not use the Python keywords and and or to combine logical array expressions.
- These keywords will test the truth value of the entire array (not element-by-element as you might expect).
- Use the bitwise operators & and | instead.

Functions	**Description**
greater(x1, x2, /[, out, where, casting, …])	Return the truth value of (x1 > x2) element-wise
greater_equal(x1, x2, /[, out, where, …])	Return the truth value of (x1 >= x2) element-wise.
less(x1, x2, /[, out, where, casting, …])	Return the truth value of (x1 < x2) element-wise.
less_equal(x1, x2, /[, out, where, casting, …])	Return the truth value of (x1 =< x2) element-wise.
not_equal(x1, x2, /[, out, where, casting, …])	Return (x1 != x2) element-wise.
equal(x1, x2, /[, out, where, casting, …])	Return (x1 == x2) element-wise.

Miscellaneous Functions

Functions	**Description**
maximum(x1, x2, /[, out, where, casting, …])	Element-wise maximum of array elements.
minimum(x1, x2, /[, out, where, casting, …])	Element-wise minimum of array elements.
Warning the behavior of maximum(a, b) is different than that of max(a, b). As a ufunc, maximum(a, b) performs an element-by-element comparison of a and b and chooses each element of the result according to which element in the two arrays is larger. In contrast, max(a, b) treats the objects a and b as a whole, looks at the (total) truth value of a > b and uses it to return either a or b (as a whole). A similar difference exists between minimum(a, b) and min(a, b).	
fmax(x1, x2, /[, out, where, casting, …])	Element-wise maximum of array elements.

fmin(x1, x2, /[, out, where, casting, …])	Element-wise minimum of array elements.

Floating Functions

- Recall that all of these functions work element-by-element over an array, returning an array output.
- The description details only a single operation.

Functions	**Description**
isfinite(x, /[, out, where, casting, order, …])	Test element-wise for finiteness (not infinity or not Not a Number).
isinf(x, /[, out, where, casting, order, …])	Test element-wise for positive or negative infinity.
isnan(x, /[, out, where, casting, order, …])	Test element-wise for NaN and return result as a boolean array.
isnat(x, /[, out, where, casting, order, …])	Test element-wise for NaT (not a time) and return result as a boolean array.
fabs(x, /[, out, where, casting, order, …])	Compute the absolute values element-wise.
signbit(x, /[, out, where, casting, order, …])	Returns element-wise True where signbit is set (less than zero).
copysign(x1, x2, /[, out, where, casting, …])	Change the sign of x1 to that of x2, element-wise.
Functions	Description
nextafter(x1, x2, /[, out, where, casting, …])	Return the next floating-point value after x1 towards x2, element-wise.
spacing(x, /[, out, where, casting, order, …])	Return the distance between x and the nearest adjacent number.
modf(x[, out1, out2], / [[, out, where, …])	Return the fractional and integral parts of an array, element-wise.
ldexp(x1, x2, /[, out, where, casting, …])	Returns x1 * 2**x2, element-wise.
frexp(x[, out1, out2], / [[, out, where, …])	Decompose the elements of x into mantissa and twos exponent.

fmod(x1, x2, /[, out, where, casting, …])	Return the element-wise remainder of division.
floor(x, /[, out, where, casting, order, …])	Return the floor of the input, element-wise.
ceil(x, /[, out, where, casting, order, …])	Return the ceiling of the input, element-wise.
trunc(x, /[, out, where, casting, order, …])	Return the truncated value of the input, element-wise.

Conditional logic

ϖ np.where is equivalent to ternary operation ‘x if condition else y’.
  - δ It takes in three parameters
    - ∇ Condition
    - ∇ X
    - ∇ Y

Mathematical and Statistical Methods

ϖ Aggregations such as sum, mean, std can be used.

Functions	Description
Sum	Sum of all the elements in the array or along an axis Zero-length arrays have sum 0.
Mean	Arithmetic mean Zero-length arrays have NaN mean
std, var	Standard deviation and variance
min, max	Minimum and maximum
argmin, argmax	Indices of minimum and maximum elements
cumsum	Cumulative sum of elements starting from 0
cumprod	Cumulative product of elements starting from 1

Methods for Boolean Arrays

ϖ sum() is used to count the True values in a boolean array.

ϖ any() is used to check if one or more values are True.

- all() is used to check if all the values are True.

Sorting

- Arrays can be sorted using the sort method.
  - <array_name>.sort()
    - a : array_like
      - Array to be sorted.
    - axis : int or None, optional
      - Axis along which to sort. If None, the array is flattened before sorting. The default is -1, which sorts along the last axis.
    - kind : {'quicksort', 'mergesort', 'heapsort', 'stable'}, optional
      - Sorting algorithm. Default is 'quicksort'.
    - order : str or list of str, optional
      - When a is an array with fields defined, this argument specifies which fields to compare first, second, etc.
      - A single field can be specified as a string, and not all fields need be specified, but unspecified fields will still be used, in the order in which they come up in the dtype, to break ties.

Set Operations

- Basic set operations can be performed on one-dimensional arrays.

Functions	Descriptions
unique(x)	Compute the sorted unique elements in x
intersect1d(x, y)	Compute the sorted common elements in x and y
union1d(x, y)	Compute the sorted union of elements
in1d(x, y)	Compute a boolean array indicating whether each element of x is contained in y
setdiff1d(x, y)	Set difference - elements in x that are not in y
setxor1d(x, y)	Set symmetric difference – elements that are in either of the arrays but not in both

Linear Algebra

Functions	Description
dot(a, b[, out])	Dot product of two arrays.

linalg.eig(a)	Compute the eigenvalues and right eigenvectors of a square array.
trace(a[, offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.
linalg.det(a)	Compute the determinant of an array.
linalg.inv(a)	Compute the (multiplicative) inverse of a matrix.
linalg.pinv(a[, rcond])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
linalg.svd(a[, full_matrices, compute_uv])	Singular Value Decomposition.
linalg.solve(a, b)	Solve a linear matrix equation, or system of linear scalar equations.

Pseudorandom Number Generation

Functions	Description
rand(d0, d1, ..., dn)	Random values in a given shape.
randn(d0, d1, ..., dn)	Return a sample/s from the "standard normal" distribution.
randint(low[, high, size, dtype])	Return random integers from low (inclusive) to high (exclusive).
random_sample([size])	Return random floats in the half-open interval [0.0, 1.0).
random([size])	Return random floats in the half-open interval [0.0, 1.0).
shuffle(x)	Modify a sequence in-place by shuffling its contents.
permutation(x)	Randomly permute a sequence, or return a permuted range
beta(a, b[, size])	Draw samples from a Beta distribution.
binomial(n, p[, size])	Draw samples from a binomial distribution.
chisquare(df[, size])	Draw samples from a chi-square distribution.
gamma(shape[, scale, size])	Draw samples from a Gamma distribution.
normal([loc, scale, size])	Draw random samples from a normal (Gaussian) distribution.
uniform([low, high, size])	Draw samples from a uniform distribution.

seed([seed])	Seed the generator.

Pandas

- It contains data structures and data manipulation tools that make data cleaning and analysis fast and easy in python.
- Pandas work on heterogenous or tabular data.
- It has to be imported as
  - import pandas as pd

Data Structures

- Series
  - It is a one dimensional array-like object containing a sequence of values and an associated array of data labels called its index.
  - The values can be of integer, string, float, python objects, etc.
    - **pandas.Series(data, index, dtype, copy)**
- DataFrames
  - It represents a rectangular table of data and contains an ordered collection of columns each of which can be a different value.
  - The DataFrame has both row and column index.
  - The data is stored as one or more two-dimensional blocks rather than a list, dict or some other collection.
    - **pandas.DataFrame(data, index, columns, dtype, copy)**
      - data : numpy ndarray (structured or homogeneous), dict, or DataFrame
        - ⇒ Dict can contain Series, arrays, constants, or list-like objects
      - index : Index or array-like
        - ⇒ Index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided
      - columns : Index or array-like
        - ⇒ Column labels to use for resulting frame. Will default to RangeIndex (0, 1, 2, …, n) if no column labels are provided
      - dtype : dtype, default None
        - ⇒ Data type to force. Only a single dtype is allowed. If None, infer
      - copy : boolean, default False
        - ⇒ Copy data from inputs. Only affects DataFrame / 2d ndarray input

Series Methods and Attributes

Attribute or Method	Description
axes	Returns a list of the row axis labels.
dtype	Returns the dtype of the object.
empty	Returns True if series is empty.
ndim	Returns the number of dimensions of the underlying data, by definition
size	Returns the number of elements in the underlying data.
values	Returns the Series as ndarray.
head()	Returns the first n rows.
tail()	Returns the last n rows.

DataFrame Attributes and Methods

Attribute or Method	Description
T	Transposes rows and columns.
axes	Returns a list with the row axis labels and column axis labels as the only members.
dtypes	Returns the dtypes in this object.
empty	True if NDFrame is entirely empty [no items]; if any of the axes are of length 0.
ndim	Number of axes / array dimensions.
shape	Returns a tuple representing the dimensionality of the DataFrame.
size	Number of elements in the NDFrame.
values	Numpy representation of NDFrame.
head()	Returns the first n rows.
tail()	Returns last n rows.

Pandas – Statistical Functions

Function	Description

count()	Number of non-null observations
sum()	Sum of values
mean()	Mean of Values
median()	Median of Values
mode()	Mode of values
std()	Standard Deviation of the Values
min()	Minimum Value
max()	Maximum Value
abs()	Absolute Value
prod()	Product of Values
cumsum()	Cumulative Sum
cumprod()	Cumulative Product

Categorical Data

- Often in real-time, data includes the text columns, which are repetitive.
  - Features like gender, country, and codes are always repetitive.
  - These are the examples for categorical data.
- Categorical variables can take on only a limited, and usually fixed number of possible values.
- Besides the fixed length, categorical data might have an order but cannot perform numerical operation.
- Categorical are a Pandas data type.
- The categorical data type is useful in the following cases
  - A string variable consisting of only a few different values.
    - Converting such a string variable to a categorical variable will save some memory.
  - The lexical order of a variable is not the same as the logical order (“one”, “two”, “three”).
    - By converting to a categorical and specifying an order on the categories, sorting and min/max will use the logical order instead of the lexical order.
  - As a signal to other python libraries that this column should be treated as a categorical variable (e.g. to use suitable statistical methods or plot types).
- Creation
  - Create a series with dtype as ‘category’
  - pandas.Categorical(values, categories, ordere`d)

- **.describe()** - Describe the type of the data
- **obj.cat.categories** - get the categories of the object.
- **obj.ordered** - get the order of the object.
- Renaming categories is done by assigning new values to the **series.cat.categories**
- **Categorical.add.categories()** - new categories can be appended.
- **Categorical.remove_categories()** - unwanted categories can be removed.

- Comparing categorical data with other objects is possible in three cases
  - comparing equality (== and !=) to a list-like object (list, Series, array, ...) of the same length as the categorical data.
    - all comparisons (==, !=, >, >=, <, and <=) of categorical data to another categorical Series, when ordered==True and the categories are the same.
    - all comparisons of a categorical data to a scalar.

Reindexing in Series

- Rearranges the data according to the new index, introducing missing values if any index values are not already present.
  - Series/dataframe.reindex(index, method, fill_value, limit, tolerance, level, copy)
    - index – new sequence to use as index.
    - method – ffill (forward fill), bfill (backward fill), nearest
    - fill_value – substitute value to be used while introducing missing data
    - limit – maximum size gap to fill
    - tolerance - maximum size gap to fill for inexact matches
    - level – match simple index on level of multiindex
    - copy – if true always copy underlying data even if new index is equivalent to old index. If false do not copy the data even when the indexes are equivalent.

Reindexing in Data frame

- Changes the row labels and column labels of a DataFrame.
- To reindex means to conform the data to match a given set of labels along a particular axis.
- Multiple operations can be accomplished through indexing like
  - Reorder the existing data to match a new set of labels.
  - Insert missing value (NA) markers in label locations where no data for the label existed.

Working with Text

Function	Description
lower()	Converts strings in the Series/Index to lower case.

upper()	Converts strings in the Series/Index to upper case.
len()	Computes String length().
strip()	Helps strip whitespace(including newline) from each string in the Series/index from both the sides.
split(' ')	Splits each string with the given pattern.
cat(sep=' ')	Concatenates the series/index elements with given separator.
get_dummies()	Returns the DataFrame with One-Hot Encoded values.
contains(pattern)	Returns a Boolean value True for each element if the substring contains in the element, else False.
replace(a,b)	Replaces the value a with the value b.
repeat(value)	Repeats each element with specified number of times.
count(pattern)	Returns count of appearance of pattern in each element.
startswith(pattern)	Returns true if the element in the Series/Index starts with the pattern.
endswith(pattern)	Returns true if the element in the Series/Index ends with the pattern.
Function	Description
find(pattern)	Returns the first position of the first occurrence of the pattern.
findall(pattern)	Returns a list of all occurrence of the pattern.
swapcase	Swaps the case lower/upper.
islower()	Checks whether all characters in each string in the Series/Index in lower case or not. Returns Boolean
isupper()	Checks whether all characters in each string in the Series/Index in upper case or not. Returns Boolean.
isnumeric()	Checks whether all characters in each string in the Series/Index are numeric. Returns Boolean.

Statistics

- pct_change() -This function compares every element with its prior element and computes the change percentage.

- cov()
  - When applied on series data computes covariance between series objects.
  - When applied on dataframe computes covariance between all the columns.
- corr() - shows the linear relationship between any two array of values (series)
- rank() - Data Ranking produces ranking for each element in the array of elements. In case of ties, assigns the mean rank.

Basic Plots with matplotlib

- Matplotlib is a python library used to create 2D graphs and plots by using python scripts.
- It has a module named pyplot which makes things easy for plotting by providing feature to control line styles, font properties, formatting axes etc.
- It supports a very wide variety of graphs and plots namely - histogram, bar charts, power spectra, error charts etc.
- It is used along with NumPy to provide an environment that is an effective open source alternative for MatLab.

Mathplotlib

- To use matplotlib it has to be imported
  - import matplotlib as plt

Simple Plot

- To create a simple plot with matplotlib the functions used are
  - plt.plot(x axis values, y axis values)
- To create the title
  - plt.title("Title")
- To define the label of X Axis
  - plt.xlabel("Label of X Axis")
- To define the label of X Axis
  - plt.ylabel("Label of Y Axis")
- To define the plot with specific colors
  - plt.plot((x, y, 'color')
- To define the line type
  - plt.plot(x, y, 'pattern')
- To save the file
  - plt.savefig('name', format='type')
- To annotate the chart by highlighting the specific locations of the chart
  - plt.annotate(text, xy)

- ∇ Text – text of the annotation
- ∇ xy – length 2 sequence specifying the (x, y) point to annotate.

ϖ To add legends to the chart

- δ plt.legend(handles, labels)

ϖ To modify the presentation style of the chart

- δ plt.style.use('style')
  - ∇ 'bmh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale', 'seaborn-bright', 'seaborn-colorblind', 'seaborn-dark-palette', 'seaborn-dark', 'seaborn-darkgrid', 'seaborn-deep', 'seaborn-muted', 'seaborn-notebook', 'seaborn-paper', 'seaborn-pastel', 'seaborn-poster', 'seaborn-talk', 'seaborn-ticks', 'seaborn-white', 'seaborn-whitegrid', 'seaborn', 'Solarize_Light2', 'tableau-colorblind10', '_classic_test'

Histogram

ϖ A histogram shows the frequency on the vertical axis and the horizontal axis is another dimension.

ϖ Usually it has bins, where every bin has a minimum and maximum value.

ϖ Each bin also has a frequency between x and infinite.

- δ n, bins, patches = plt.hist(x, bins, facecolor, alpha, rwidth)

import numpy as np	**import numpy as np**
a = np.array([1,2,3])	from numpy.core.multiarray import ndarray
print(a)	
print(type(a))	a = np.array([1,2,3])
	print(a)
a = np.array([1,2,3],dtype=complex)	
print(a)	a= np.array([(1,2,3),(4,5,6)])
print(type(a))	print(a)
a = np.asanyarray([1,2,3])	a = np.array([1, 2, 3,4,5], ndmin = 2)
print(a)	print(a)
print(type(a))	
	a = np.array((1, 2, 3), dtype = complex)
a= np.arange(1, 10, 2)	print(a)
print(a)	
print(type(a))	list2 = [[0,1,2], [3,4,5], [6,7,8]]
	arr2d = np.array(list2)
a=np.ones(2)	print(arr2d)
print(a)	
print(type(a))	arr2d_f = np.array(list2, dtype='float')
	print(arr2d_f)
l=[(1,2,3), (1,2,3)]	
a= np.ones_like(l)	arr = arr2d_f.astype('int')
print(a)	print(arr)

print(type(a))	arr1 = arr2d_f.astype('int').astype('str')
	print(arr1)
a=np.zeros(2)	arr2d_b = np.array([1, 0, 10], dtype='bool')
print(a)	print(arr2d_b)
print(type(a))	print(type(arr2d_b))
l=[(1,2,3), (1,2,3)]	arr1d_obj = np.array([1, 'a'], dtype='object')
a= np.zeros_like(l)	print(arr1d_obj)
print(a)	print(type(arr1d_obj))
print(type(a))	
	arr1d_obj = arr1d_obj.tolist()
a=np.empty(2)	print(arr1d_obj)
print(a)	print(type(arr1d_obj))
print(type(a))	
	import numpy as np
a=np.empty([2,2])	array = np.arange(8)
print(a)	print("Original array : \n", array)
print(type(a))	
	# shape array with 2 rows and 4 columns
l=[(1,2,3), (1,2,3)]	array = np.arange(8).reshape(2, 4)
a=np.empty_like(l)	print("\narray reshaped with 2 rows and 4 columns : \n", array)
print(a)	
print(type(a))	# shape array with 2 rows and 4 columns
	array = np.arange(8).reshape(4, 2)
a=np.full(2,10)	print("\narray reshaped with 2 rows and 4 columns : \n", array)
print(a)	
print(type(a))	# Constructs 3D array
	array = np.arange(8).reshape(2, 2, 2)
l=[(1,2,3), (1,2,3)]	print("\nOriginal array reshaped to 3D : \n", array)
a=np.full_like(l,10)	
print(a)	ar = np.arange(8)
print(type(a))	print(ar)
	print(np.shape(ar))
a=np.eye(2)	print(np.ndim(ar))
print(a)	print(ar.dtype)
print(type(a))	print(ar.size)
a=np.identity(2)	**import numpy as np**
print(a)	arr = np.arange(15).reshape((3,5))
print(type(a))	print(arr)
	print(arr.T)
import numpy as np	print(np.dot(arr, arr.T))
arr = np.random.randn(100)	arr=np.arange(16).reshape((2,2,4))
	print(arr)
sumofpos = (arr>0).sum()	print(arr.transpose((1,0,2)))

sumofneg = (arr<0).sum()	
	arr= np.array([[0,1,2],[3,4,5],[6,7,8]])
print(arr)	print(arr)
print("The number of positive numbers is ", sumofpos)	print(arr.transpose(1,0))
print("The number of negative numbers is ", sumofneg)	
	import numpy as np
boolarr = np.where((arr<0), True, False)	'''Solve the system of equations 3 * x0 + x1 = 9 and x0 + 2 * x1 = 8'''
print(boolarr)	a = np.array([[3,1], [1,2]])
print(boolarr.any())	b = np.array([9,8])
print(boolarr.all())	x = np.linalg.solve(a, b)
	print(a,b,x)
import numpy as np	
import pandas as pd	
print("Reindexing Series")	print("Dot product of 3 and 4 is ",np.dot(3, 4))
obj = pd.Series([4.5, 6.7, -1.2, 3.4], index = ['d','a','b','c'])	print("Dot product of complex numbers is ", np.dot([2j, 3j], [2j, 3j]))
print(obj)	a = [[1, 0], [0, 1]]
obj1=obj.reindex(['a', 'b', 'c', 'd', 'e'])	b = [[4, 1], [2, 2]]
print(obj1)	print(a, b, np.dot(a, b))
obj2= pd.Series(['blue', 'green', 'yellow'], index = [0, 2, 4])	
print(obj2)	a = np.arange(3*4*5*6).reshape((3,4,5,6))
print(obj2.reindex(range(6), method = 'ffill'))	b = np.arange(3*4*5*6)[::-1].reshape((5,4,6,3))
	print(a, b, np.dot(a, b)[2,3,2,1,2,2])
obj2= pd.Series(['blue', 'green', 'yellow'], index = [0, 3, 7])	
print(obj2)	s=sum(a[2,3,2,:] * b[1,2,:,2])
print(obj2.reindex(range(9), method = 'nearest'))	print(s)
	arr=np.array([[1,2,3],[4,5,6],[7,8,9]])
print("Reindexing DataFrame")	print(arr)
df = pd.DataFrame(np.arange(9).reshape((3, 3)), index=['a','c','d'], columns = ['Bangalore', 'Mysore', 'Mangalore'])	print(np.trace(arr))
print(df)	
df1= df.reindex(['a','b','c','d'], fill_value=2)	**import numpy as np**
print(df1)	print("Rand function is ")
cities = ['Bangalore', 'Mangalore', 'Mysore']	print(np.random.rand(3,2))
print(df.reindex(columns = cities))	
	print("Randn function is ")
import numpy as np	print(np.random.randn())
names = np.array(['Bob', 'Joe', 'will', 'Bob', 'Joe', 'Will','Doe','Jane'])	print(2.5 * np.random.randn(2, 4) + 3)
print(names)	
print("The unique values are ")	print("Randint function is ")
print(np.unique(names))	print(np.random.randint(2, size=10))

	print(np.random.randint(3, size=10))
ints = np.array([[3,2,1], [2,3,8],[9,0,8]])	print(np.random.randint(5, size=10))
print(ints)	print(np.random.randint(5, size=(2, 4)))
print("The unique values are ")	
print(np.unique(ints))	print("Random Sample function is ")
ints2 = np.array([[1,2,1], [2,3,4],[9,0,6]])	print(np.random.random_sample())
print("The intersect values are ")	print(np.random.random_sample((5,)))
print(np.intersect1d(ints, ints2))	'''Three-by-two array of random numbers from [-5, 0)'''
	print(5 * np.random.random_sample((3, 2)) - 5)
print("The union of the arrays are ")	
print(np.union1d(ints, ints2))	arr = np.arange(10)
	print(arr)
print("The set difference of the arrays is")	print("The shuffled array is ")
print(np.setdiff1d(ints, ints2))	np.random.shuffle(arr)
	print(arr)
print("The symmetric difference of the arrays is")	
print(np.setxor1d(ints, ints2))	arr = np.arange(10)
	print(arr)
print("The elements of ints present in the ints2 arrays is")	print("The permuted array is ")
print(np.in1d(ints, ints2))	print(np.random.permutation(10))
import numpy as np	print(np.random.permutation([1, 4, 9, 12, 15]))
arr = np.random.randn(6)	
print(arr)	arr = np.arange(9).reshape((3, 3))
print("The sorted array is ")	print(arr)
arr.sort()	print(np.random.permutation(arr))
print(arr)	
	n, p = 10, .5 # number of trials, probability of each trial
arr = np.random.randn(5,3)	s = np.random.binomial(n, p, 1000)
print(arr)	print(s)
arr.sort()	
print("The sorted array is ")	sum=sum(np.random.binomial(9, 0.1, 20000) == 0)/20000.
print(arr)	print(sum)
arr = np.array([[1,3,2,4],[6,2,4,1]])	print("Chisquare")
print(arr)	print(np.random.chisquare(2, 4))
arr.sort(axis=0)	
print("The sorted array is ")	
print(arr)	shape, scale = 2., 2. # mean=4, std=2*sqrt(2)
	s = np.random.gamma(shape, scale, 1000)
arr = np.array([[1,3,2,4],[6,5,4,8]])	print("Gamma DIstribution")
print(arr)	print(s)
arr.sort(axis=1)	
print("The sorted array is ")	mu, sigma = 0, 0.1 # mean and standard deviation
print(arr)	s = np.random.normal(mu, sigma, 1000)
	print("Normal Distribution")
	print(s)

dtype = [('name', 'S10'), ('height', float), ('age', int)]	print("Uniform Distribution")
values = [('Arthur', 1.8, 41), ('Lancelot', 1.9, 38), ('Galahad', 1.7, 38)]	s = np.random.uniform(-1,0,1000)
a = np.array(values, dtype=dtype) # create a structured array	print(s)
print(a)	print(np.all(s >= -1))
print("Sorting based on height")	print(np.all(s < 0))
print(np.sort(a, order='height'))	
print("Sorting based on age and if age is equal then based on height")	**import numpy as np**
print(np.sort(a, order=['age', 'height']))	xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
	yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
import numpy as np	cond = np.array([True, False, True, True, False])
arr = np.arange(16).reshape((2,2,4))	result = np.where(cond, xarr, yarr)
print(arr)	''' Take a value from xarr whenever the corresponding value in cond is True and value from yarr whenever it is false'''
print(arr.swapaxes(1,2))	print(result)
	print(type(result))
import numpy as np	
a = np.array([[0,1,2],[3,4,5], [6,7,8]])	arr = np.random.randn(4,4)
print(a)	print(arr)
print("First two columns of all rows", a[:3, :2])	''' replace all positive values with 2 and all negative values with -2'''
print(a[:3, 2])	res = np.where(arr>0, 2, -2)
print(a[2:])	print(res)
print(a[1:, 1])	
print(a[1,:2])	''' replace all positive values with 2 '''
	res = np.where(arr>0, 2, arr)
array = np.array(['Alpha', 'Beta', 'Gamma',	print(res)
'Alpha', 'Omega', 'Pi', 'Beta', 'Alpha', 'Delta'])	
print(array)	**import numpy as np**
print(array[array=='Alpha'])	arr = np.random.randn(5,4)
data = np.random.randn(9, 4)	print(arr)
print(data)	print("Sum of all elements in array is ", arr.sum())
print(data[array=='Alpha'])	print("Mean of the array is ", arr.mean())
print(data[array=='Alpha', 2:])	
print(data[array=='Alpha', 1:])	print("Sum of all elements in array with axis 0 is ", arr.sum(axis=0))
print(data[~(array=='Alpha')])	print("Mean of the array is with axis 0 is ", arr.mean(axis=0))
c=array=='Beta'	
print(type(c))	print("Sum of all elements in array with axis 1 is ", arr.sum(axis=1))
print(data[c])	print("Mean of the array is with axis 1 is ", arr.mean(axis=1))
print(data[~c])	
print(data[array!='Alpha'])	arr=np.array([0,1,2,3,4,5,6,7])
c = (array=='Alpha') | (array=='Gama')	print(arr)

print(data[c])	print("The cummulative sum is ", arr.cumsum())
print(type(c))	print("The cummulative sum across axis 0 is ", arr.cumsum(axis = 0))
import pandas as pd	arr=np.array([[0,1,2],[3,4,5],[6,7,8]])
data = {'city':['Bangalore', 'Bangalore', 'Bangalore', 'Mangalore', 'Mangalore', 'Mangalore'],	print(arr)
'year':[1991, 2001, 2011, 1991, 2001, 2011],	print("The cummulative sum is ")
'population': [4839162, 6537124, 9621551, 1656165, 1897730	print(arr.cumsum())
frame = pd.DataFrame(data)	print("The cummulative sum across axis 0 is ")
print("Data Frame from Dictionary")	print(arr.cumsum(axis = 0))
print(frame)	print("The cummulative sum across axis 1 is ")
print("Data Frame with headers")	print(arr.cumsum(axis = 1))
print(frame.head())	
print("Data Frames with specific columns")	arr=np.array([[1,2,3],[4,5,6],[7,8,9]])
print(pd.DataFrame(data, columns = ['year', 'city', 'population']))	print(arr)
	print("The cummulative product is ")
print("DataFrames created with lists")	print(arr.cumprod())
data = [1,2,3,4,5]	print("The cummulative product across axis 0 is ")
df = pd.DataFrame(data)	print(arr.cumprod(axis = 0))
print(df)	print("The cummulative product across axis 1 is ")
	print(arr.cumprod(axis = 1))
data = [['Alex',10],['Bob',12],['Clarke',13]]	
df = pd.DataFrame(data,columns=['Name','Age'])	arr=np.array([[0,1,2],[3,4,5],[6,7,8]])
print(df)	print(arr)
	print("The standard deviation is ")
data = [['Alex',10],['Bob',12],['Clarke',13]]	print(arr.std())
df = pd.DataFrame(data,columns=['Name','Age'],dtype=float)	print("The Variation is ")
print(df)	print(arr.var())
print("DataFrames from a List of Dictionary")	print("The standard deviation across axis 0 is ")
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]	print(arr.std(axis = 0))
df = pd.DataFrame(data)	print("The Variation across axis 0 is ")
print(df)	print(arr.var(axis = 0))
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]	print("The standard deviation across axis 1 is ")
	print(arr.std(axis = 1))
#With two column indices, values same as dictionary keys	print("The Variation across axis 1 is ")
df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b'])	print(arr.var(axis = 1))
#With two column indices with one index with other name	arr=np.array([[0,1,2],[3,4,5],[6,7,8]])

df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b1'])	print(arr)
print(df1)	print("The maximum is ")
print(df2)	print(arr.max())
	print("The minimum is ")
print("DataFrames from Dictionary of Series")	print(arr.min())
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),	
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}	print("The index of maximum element is ")
	print(arr.argmax())
df = pd.DataFrame(d)	print("The index of minimum element is ")
print(df)	print(arr.argmin())
print("DataFrame with Column Selection")	print("The maximum across axis 0 is ")
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),	print(arr.max(axis = 0))
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}	print("The minimum across axis 0 is ")
	print(arr.min(axis = 0))
df = pd.DataFrame(d)	
print(df ['one'])	print("The index of maximum element across axis 0 is ")
	print(arr.argmax(axis = 0))
print("DataFrame with Column Addition")	print("The index of minimum element across axis 0 is ")
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),	print(arr.argmin(axis = 0))
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}	
	print("The maximum across axis 1 is ")
df = pd.DataFrame(d)	print(arr.max(axis = 1))
	print("The minimum across axis 1 is ")
# Adding a new column to an existing DataFrame object with column label by passing new series	print(arr.min(axis = 1))
print ("Adding a new column by passing as Series:")	print("The index of maximum element across axis 1 is ")
df['three']=pd.Series([10,20,30],index=['a','b','c'])	print(arr.argmax(axis = 1))
print(df)	print("The index of minimum element across axis 1 is ")
	print(arr.argmin(axis = 1))
print ("Adding a new column using the existing columns in DataFrame:")	
df['four']=df['one']+df['three']	**import pandas as pd**
print(df)	**import numpy as np**
	print("AXES")
print("DataFrame with Column Deletion")	df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),	print(df.axes)
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd']),	
'three' : pd.Series([10,20,30], index=['a','b','c'])}	print("DATA TYPES")
	df = pd.DataFrame({'float': [1.0], 'int': [1],'datetime': [pd.Timestamp('20180310')], 'string': ['foo']})
df = pd.DataFrame(d)	print(df.dtypes)
print ("Our dataframe is:")	

print(df)	print("CHECK IF DATAFRAME IS EMPTY")
	df_empty = pd.DataFrame({'A' : []})
# using del function	print(df_empty)
print ("Deleting the first column using DEL function:")	print(df_empty.empty)
del df['one']	
print(df)	print("DIMENSIONS")
	s = pd.Series({'a': 1, 'b': 2, 'c': 3})
# using pop function	print(s.ndim)
print ("Deleting another column using POP function:")	df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
df.pop('two')	print(df.ndim)
print(df)	
	print("SIZE")
print("DataFrame with Row Selection")	s = pd.Series({'a': 1, 'b': 2, 'c': 3})
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),	print(s.size)
'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}	df = pd.DataFrame({'col1': [1, 2], 'col2': [3, 4]})
	print(df.size)
df = pd.DataFrame(d)	
print(df.loc['b'])	print("VALUES")
	df = pd.DataFrame({'age': [3, 29], 'height': [94, 170],'weight': [31, 115]})
df = pd.DataFrame(d)	print(df.values)
print(df.iloc[2])	df2 = pd.DataFrame([('parrot', 24.0, 'second'),('lion',80.5, 1),('monkey', np.nan, None)],columns=('name', 'max_speed', 'rank'))
	print(df2.values)
print("Sliced Rows")	
df = pd.DataFrame(d)	print("HEAD")
print(df[2:4])	df = pd.DataFrame({'animal':['alligator', 'bee', 'falcon', 'lion','monkey', 'parrot', 'shark', 'whale', 'zebra']})
	print(df.head())
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])	print("TAIL")
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])	print(df.tail())
print(df)	
df = df.append(df2)	**import pandas as pd**
print("After Appending")	**import numpy as np**
print(df)	
	df = pd.DataFrame({"Person":["John", "Myla", None, "John", "Myla"], "Age": [24., np.nan, 21., 33, 26],"Dependents": [False, True, True, True, False]})
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])	print(df)
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])	print("COUNT")
print(df)	print(df.count())
df = df.append(df2)	print("COUNTS FOR EACH ROW")
print("After appending")	print(df.count(axis='columns'))
print(df)	print("COUNTS FOR ONE LEVEL OF A MULTIINDEX")
# Drop rows with label 0	print(df.set_index(["Person", "Dependents"]).count(level="Person"))
df = df.drop(0)	

print("After dropping rows with label 0")	print("SUM")
print(df)	print("SUM OF EMPTY SERIES")
	print(pd.Series([]).sum())
data = {'city':['Bangalore', 'Bangalore', 'Bangalore', 'Mangalore', 'Mangalore', 'Mangalore'],	print("SUM OF SERIES WITH MIN COUNT ")
'year':[1991, 2001, 2011, 1991, 2001, 2011],	print(pd.Series([]).sum(min_count=1))
'population': [4839162, 6537124, 9621551, 1656165, 1897730	print("SUM ROW WISE")
frame = pd.DataFrame(data)	print(df.sum(axis = 'rows'))
print(frame)	print("SUM OF SKIPPED NA VALUES")
print("Rearrange the dataframe")	print(pd.Series([np.nan]).sum())
frame = pd.DataFrame(data, columns = ['year','city','population'])	print(pd.Series([np.nan]).sum(min_count=1))
print(frame)	
	d = {
print("Data Frame with indexes")	'Name': ['Alisa', 'Bobby', 'Cathrine', 'Madonna', 'Rocky', 'Sebastian', 'Jaqluine',
frame = pd.DataFrame(data, columns =['year','city','population'], index = ['one', 'two', 'three', 'four', 'five', 'six'])	'Rahul', 'David', 'Andrew', 'Ajay', 'Teresa'],
print(frame)	'Score1': [62, 47, 55, 74, 31, 77, 85, 63, 42, 32, 71, 57],
	'Score2': [89, 87, 67, 55, 47, 72, 76, 79, 44, 92, 99, 69],
print("Data Frame with specific location")	'Score3': [56, 86, 77, 45, 73, 62, 74, 89, 71, 67, 97, 68],
print(frame.loc['three'])	'Score4': [79, 57, 67, 85, 47, 72, 76, 79, 44, 92, 100, 69],
	'Score5': [46, 46, 77, 35, 73, 62, 74, 89, 71, 67, 96, 68],
print("Summary of the data frame")	
print(frame.describe())	}
	df = pd.DataFrame(d)
import pandas as pd	print(df)
import numpy as np	print("SUM")
	print(df.sum())
print("Pandas Series with system defined index")	print("SUM COLUMN WISE")
	print(df.sum(axis=0))
obj = pd.Series([4, 2, 3, -3])	print("SUM ROW WISE")
print(obj)	print(df.sum(axis=1))
print(type(obj))	print("SPECIFIC COLUMN SUM")
print("Series values")	print(df.loc[:,"Score1"].sum())
print(obj.values)	print(df.loc[:,"Score2"].sum())
print("Series index")	print(df.loc[:,"Score3"].sum())
print(obj.index)	print(df.loc[:,"Score4"].sum())
print("Individual series elements")	print(df.loc[:,"Score5"].sum())
print(obj[2])	

print("Pandas Series with user defined index")	
obj = pd.Series([4, 2, 3, -3], index =['d','b','c','a'])	df = pd.DataFrame(d)
print(obj)	print(df)
print(type(obj))	print("MEAN")
print("Series values")	print(df.mean())
print(obj.values)	print("MEAN COLUMN WISE")
print("Series index")	print(df.mean(axis=0))
print(obj.index)	print("MEAN ROW WISE")
print("Individual series elements")	print(df.mean(axis=1))
print(obj['c'])	print("SPECIFIC COLUMN MEAN")
	print(df.loc[:,"Score1"].mean())
print("Operations on Series")	print(df.loc[:,"Score2"].mean())
print("Positive Values")	print(df.loc[:,"Score3"].mean())
print(obj[obj>0])	
print("Product of the series with 2")	print("MEDIAN")
print(obj*2)	print(df.median())
print("Exponential value of the series")	print("MEDIAN COLUMN WISE")
print(np.exp(obj))	print(df.median(axis=0))
print("Check if a particular index exists")	print("MEDIAN ROW WISE")
print('b' in obj)	print(df.median(axis=1))
print('e' in obj)	print("SPECIFIC COLUMN MEDIAN")
	print(df.loc[:,"Score1"].median())
	print(df.loc[:,"Score2"].median())
sdata ={'Bangalore': 5600000, 'Mysore': 24000000, 'Mangalore':3400000, 'Hassan': 8989898}	print(df.loc[:,"Score3"].median())
obj = pd.Series(sdata)	
print("Dictionary as Panda Series")	print("MODE")
print(obj)	print(df.mode())
print(type(obj))	print("MODE COLUMN WISE")
	print(df.mode(axis=0))
cities = ['Chikamagalur', 'Bangalore', 'Mangalore', 'Mysore']	print("MODE ROW WISE")
obj=pd.Series(sdata, index = cities)	print(df.mode(axis=1))
print(obj)	print("SPECIFIC COLUMN MODE")
	print(df.loc[:,"Score1"].mode())
import numpy as np	print(df.loc[:,"Score2"].mode())
import pandas as pd	print(df.loc[:,"Score3"].mode())
s = pd.Series(np.random.randn(10))	print("STANDARD DEVIATION")
print("Series is ",s)	print(df.std())
print("Min is ",s.agg('min'))	print("STD COLUMN WISE")
print("Max is ", s.agg('max'))	print(df.std(axis=0))
print("Mean is ", s.agg('mean'))	print("STD ROW WISE")
print("Median is ", s.agg('median'))	print(df.std(axis=1))
print("Sum is ", s.agg('sum'))	print("SPECIFIC COLUMN STD")
print("Product is ", s.agg('prod'))	print(df.loc[:,"Score1"].std())
print("Standard Deviation is ", s.agg('std'))	print(df.loc[:,"Score2"].std())
print("Variance is ", s.agg('var'))	print(df.loc[:,"Score3"].std())

	d = {
	'Name': ['Alisa', 'Bobby', 'jodha', 'jack', 'raghu', 'Cathrine',
	'Alisa', 'Bobby', 'kumar', 'Alisa', 'Alex', 'Cathrine'],
	'Age': [26, 24, 23, 22, 23, 24, 26, 24, 22, 23, 24, 24],
	'Score': [85, 63, 55, 74, 31, 77, 85, 63, 42, 62, 89, 77]}
	df = pd.DataFrame(d, columns=['Name', 'Age', 'Score'])
	print(df)
	print("MINIMUM")
	print(df.min())
	print("MINIMUM AGE")
	print(df['Age'].min())
	print("MINIMUM NAME")
	print(df['Name'].min())
	print("MAXIMUM")
	print(df.max())
	print("MAXIMUM AGE")
	print(df['Age'].max())
	print("MAXIMUM NAME")
	print(df['Name'].max())
	df = pd.DataFrame({'a': [4, 5, 6, 7],'b': [10, 20, 30, 40],'c': [100, 50, -30, -50]})
	print(df)
	print("ABSOLUTE VALUES")
	print(df.abs())
	print("PRODUCT VALUES")
	print(df.prod())
	df = pd.DataFrame([[2.0, 1.0],[3.0, np.nan],[1.0, 0.0]],columns=list('AB'))
	print("CUMMULATIVE SUM")
	print(df.cumsum())
	print("CUMMULATIVE PRODUCT")
	print(df.cumprod())
	import pandas as pd
	import numpy as np
	df = pd.DataFrame({"Person":["John", "Myla", None, "John", "Myla"], "Age": [24., np.nan, 21., 33, 26],"Dependents": [False, True, True, True, False]})
	print(df)

	print("COUNT")

Dr. Lekha A | Associate Professor, Department of Computer Applications, PESU