FUNDAMENTALS OF DATA SCIENCE

Data science: Definition, Datafication, Exploratory Data Analysis, The Data science process, A data scientist role in this process.

NumPy Basics: The NumPy ndarray: A Multidimensional Array Object, Creating ndarrays ,Data Types for ndarrays, Operations between Arrays and Scalars, Basic Indexing and Slicing, Boolean Indexing, Fancy Indexing, Data Processing Using Arrays, Expressing Conditional Logic as Array Operations, Methods for Boolean Arrays, Sorting, Unique.

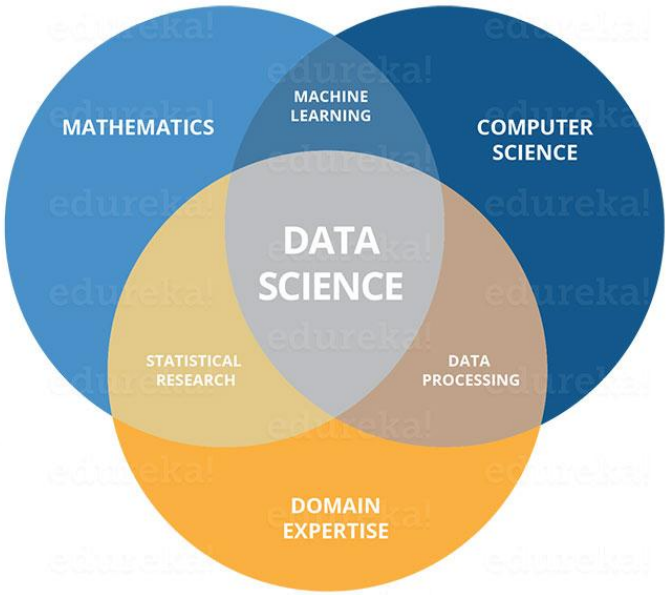
Data: It is meaningful raw fact which can be processed and stored as information.

Information: It is collection of data which provides a meaning statement.

Data science: The study of data. It involves developing methods of storing, recording and analyzing of any type of data [structured, unstructured, and semi structured].

Goal of data science: The main goal of data science is for gain clear insight and knowledge of various types of data to extract useful information.

- To gain insight and knowledge of data, it uses scientific approaches and procedures, algorithms, methodologies and framework



Job Roles of Data Science:

- Data scientist
- Decision scientist
- Data architect

- Data manager
- Machine learning manager
- Statistician
- Data science engineer

Datafication: It is a process of collection "all aspects of life" and converting them into data for making new opportunities and new challenges.

(Or)

It refers to a process by which the subject, object and practices are transformed into digital data, that makes rise of digital technologies digitalization and Big data.

Ex: LinkedIn data files professional data, sancition/rejection of loan.

Need of Datafication:

- One the process has converted into data ,they can filly tracked, monitored and ultimately optimized
- Datafication enables the organization to operate in a great effective manner with good decision making.

Big Data:

Big data is a term used to describe a collection of data i.e. huge in size and yet growing exponentially with time.

(Or)

Big Data is a collection of data that is huge in volume, yet growing exponentially with time. It is a data with so large size and complexity that none of traditional data management tools can store it or process it efficiently. Big data is also a data but with huge size.

Ex:

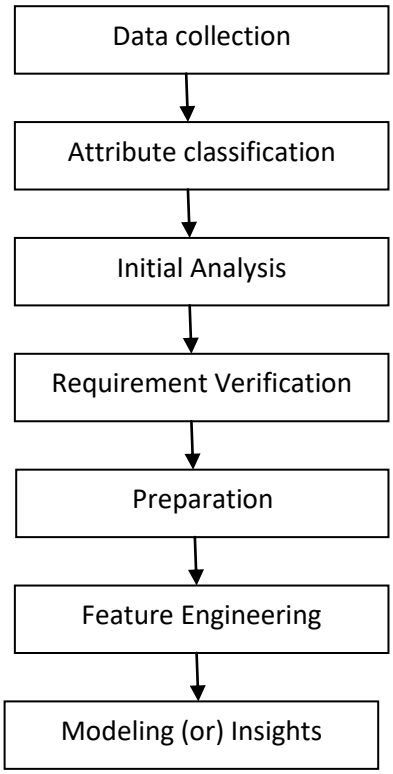
1. The New York Stock Exchange is an example of Big Data that generates about *one terabyte* of new trade data per day.
2. The statistic shows that 500+terabytes of new data get ingested into the databases of social media site Facebook, every day. This data is mainly generated in terms of photo and video uploads, message exchanges, putting comments etc.

Exploratory data analysis:

- It is a part of data science process.
- Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods.

- The main purpose of EDA is to help look at data before making any assumptions. It can help identify obvious errors, as well as better understand patterns within the data, detect outliers or anomalous events, find interesting relations among the variables.

Block Diagram of Exploratory data analysis:



- **Data collection:** Initially data is collected for analysis.
- **Attribute classification:** Attributes are the descriptive properties which are owned by each entity of an entity set .In the classification of attribute, it shows the data types, dimension and the metadata.
- **Initial Analysis:** Here in initial analysis is associated with the following
  Univariant: Univariate analysis is the simplest form of analyzing data. "Uni" means "one", so in other words your data has only one variable. It doesn't deal with causes or relationships (unlike regression) and it's major purpose is to describe; It takes data, summarizes that data and finds patterns in the data.
  Multivariate: Multivariate analysis (**MVA**) is a Statistical procedure for analysis of data involving more than one type of measurement or observation. It may also mean solving problems where more than one dependent variable is analyzed simultaneously with other variables.

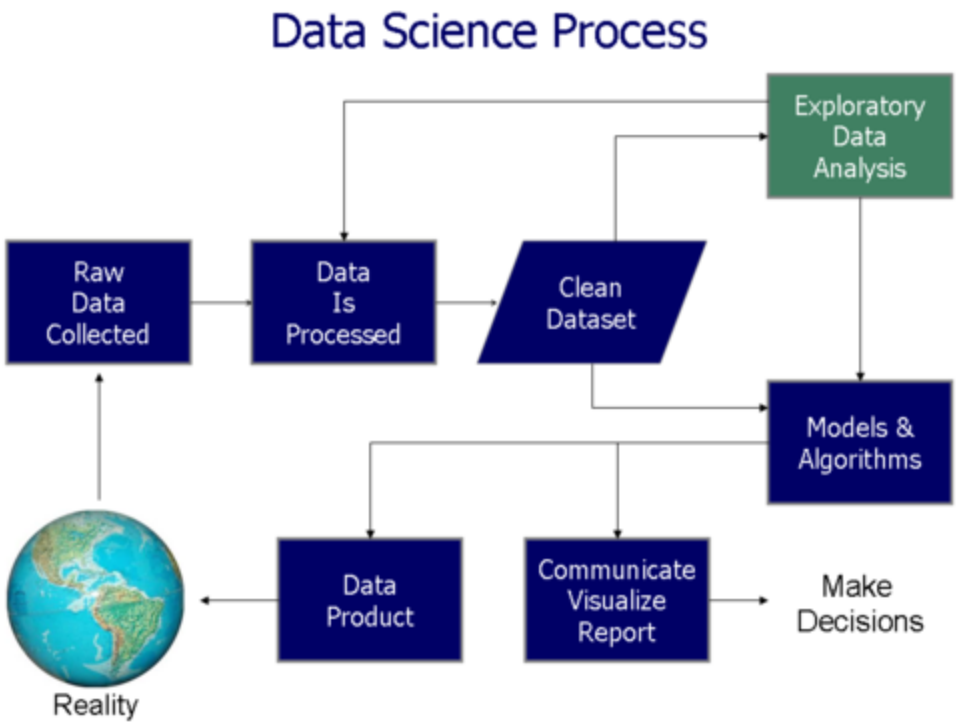
Analomy Detection: It is the process of identifying unexpected items or events in data sets, which differ from the norm. And anomaly detection is often applied on unlabeled data which is known as unsupervised anomaly detection.

- **Requirement Verification:** In this step EDA will check all requirement it needed from the next process.
- **Preparation:** In this step of preparation of cleaning involves the following steps
  Data scaling: It is a method used to normalize the range of independent variables or features of data. In data processing, it is also known as data normalization and is generally performed during the data preprocessing step.
  Data cleaning: Data cleaning is the process of fixing or removing incorrect, corrupted, incorrectly formatted, duplicate, or incomplete data within a dataset. When combining multiple data sources, there are many opportunities for data to be duplicated or mislabeled. If data is incorrect, outcomes and algorithms are unreliable, even though they may look correct.
  Data wrangling: Data wrangling is the process of cleaning and unifying messy and complex data sets for easy access and analysis. This process typically includes manually converting and mapping data from one raw form into another format to allow for more convenient consumption and organization of the data.
- **Feature Engineering:** This feature engineering contains the data extraction process.
  Data Extraction: Data extraction is the process of collecting or retrieving disparate types of data from a variety of sources, many of which may be poorly organized or completely unstructured. Data Extraction makes it possible to consolidate, process and refine data so that it can be stored in a centralized location in order to be transformed.
- **Modeling (or) Insight:** Here actual machine learning and deep learning comes into picture in order to model the data using several machine learning algorithm.

Data science process: Data Science is all about a systematic process used by Data Scientists to analyze, visualize and model large amounts of data. A data science process helps data scientists use the tools to find unseen patterns, extract data, and convert information to actionable insights that can be meaningful to the company.

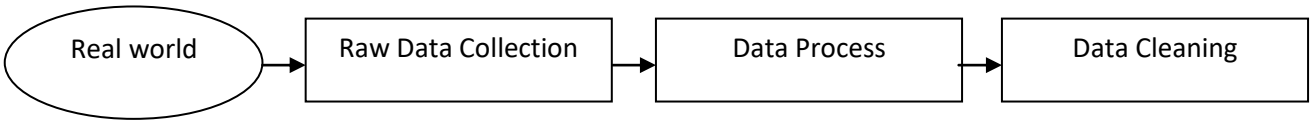
Data Modeling: Data modeling is the process of producing a descriptive diagram of relationships between various types of information that are to be stored in a database. One of the goals of data modeling is to create the most efficient method of storing information while still providing for complete access and reporting.

Data science process visualization:



1. First, we have a real world where the people are busy with various activities like browsing, shopping, video games.
2. Initially we need to collect the raw data (Metadata (or) catalog) i.e. collection data about what data should be collected.
3. Now we need to process this data to make it clean for analysis, so we built and use pipelines for data munging (it includes data joining, data wrangling, and data scrapping).
4. Once if we have clean data then we should be doing analysis called exploratory data analysis (EDA).
5. In the process of EDA, if you realize that it is not actually clean(because of missing values, duplicates , multivalues, outlayers).Again we need to collect the raw data and repeat the process.
6. If you satisfies with EDA then we need to start building the model by using different machine learning algorithms like recursion, KNN (K nearest neighbor), decision tree algorithm, etc.
7. Alternatively we are going to build a data product where we can throw back to the real world, where it can generate more data that can create feedback loop.

Role of the Data Scientist in data process:



- A data scientist's role combines computer science, statistics, and mathematics. They analyze, process, and model data then interpret the results to create actionable plans for companies and other organizations.
- Data scientists are analytical experts who utilize their skills in both technology and social science to find trends and manage data. They use industry knowledge, contextual understanding, and skepticism of existing assumptions – to uncover solutions to business challenges.

From the block diagram we can say that the data scientist start the data process by collecting the raw data from the real world. Then it will be processed for the data cleaning

Ex: Assume scientist as the real world such that raw data will be the experimental values of their experiments, then it will undergo some data process (data munging, data joining, data scrapping).After the data processing , data cleaning is performed(for debugging, missing values and cleaning).

Numpy-Ndarray:

NumPy is the fundamental package for scientific computing in Python. It stands for numerical python It is a Python library that provides a multidimensional array object, various derived objects and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. **Travis Oliphant** created NumPy package in 2005 by injecting the features of the ancestor module Numeric into another module Numarray.

It provides supports for large multi-dimensional array object and various tools to work with them. Various other libraries like pandas, matplotlib, scikit-learn are built top of this amazing library.

Arrays are the collection of elements/values that can have one (or) more dimensions. An array of one dimension is called vector and array of two dimensions is called a matrix. Numpy arrays are called ndarray (or) n-dimensional array. They store the elements of the same type and provides efficient storage.

The need of numpy:

Numpy provide a easy and efficient way to handle huge amount of data. Numpy is also very convenient with Matrix multiplication and data reshaping. NumPy is fast which makes it reasonable to work with a large set of data.

Advantages of numpy:

1. Numpy performs array oriented programs.
2. It is efficient to implement the multi-dimensional arrays.
3. It performs scientific calculation.
4. NumPy provides the in-built functions for linear algebra and random number generation.

NumPy in combination with SciPy and Matplotlib is used as the replacement to MATLAB as Python is more complete and easier programming language than MATLAB.

Creating of numpy array:

We can create a NumPy ndarray object by using the "array ()" function. To create an ndarray, we can pass a list, tuple or any array-like object into the array () method and it will be converted into an ndarray.

Basic ndarray:

```
import numpy as np

a=np.array([[1,2,3,4,5],[6,7,8,9,10],[11,12,13,14,15]])

print (a)
```

Output: [[1 2 3 4 5]

[6 7 8 9 10]

[11 12 13 14 15]]

Array of zeros:

This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 0.

Syntax: numpy. zeros (shape)

```
import numpy as np

b=np.zeros(5)

print(b)
```

output:[0 0 0 0 0]

Array of ones:

It is used to create the numpy array with the specified shape where each numpy array item is initialized to 1.

Syntax: np.ones(shape)

```
import numpy as np

d=np.ones(6)

print(d)
```

output:[1 1 1 1 1 1]

Array of random values:

The random is a module present in the NumPy library. This module contains the functions which are used for generating random numbers. This module contains some simple random data generation methods, some permutation and distribution functions, and random generator functions.

Syntax: np.random.rand(shape)

```
import numpy as np

e=np.random.rand(6)

print(e)
```

output: [0.02905376 0.59423152 0.25030791 0.60751057 0.52254074 0.80428618]

Array of your choice:

To print the array elements of your choice we use full().

```
import numpy as np

f=np.full((3,3),7)

print(f)
```

output:[7 7 7]

[7 7 7]

[7 7 7]

Imatrix with numpy:

For printing identity matrix we use eye().

Syntax :np.eye(size)

```
import numpy as np
g=np.eye(4)
print(g)
```

output: [1 0 0 0]

[0 1 0 0]

[0 0 1 0]

[0 0 0 1]

Evenly spaced ndarray:

It creates an array by using the evenly spaced values over the given interval. The syntax to use the function is given below.

Syntax: np.arrange(start, stop, step, dtype)

It accepts the following parameters.

1. **start:** The starting of an interval. The default is 0.
2. **stop:** represents the value at which the interval ends excluding this value.
3. **step:** The number by which the interval values change.
4. **dtype:** the data type of the numpy array items.

```
import numpy as np
h = np.arange(0,10,2,float)
print(h)
```

output: [0 2 4 6 8]

dtype: Every ndarray has an associated data type (dtype) object. This data type object (dtype) informs us about the layout of the array.

```
import numpy as np

a=np.array([1,2,3])
```

print(a.dtype)

output: int 16

Shape of the given ndarray:

The shape of an array can be defined as the number of elements in each dimension. Dimension is the number of indices or subscripts, that we require in order to specify an individual element of an array.

Syntax: *numpy.shape(array_name)*

```
import numpy as np

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

print(a.shape)
```

output:(2,3)

Dimension of the given array:

To find the dimensions of the given array we use ndim function. It will return the dimension.

```
import numpy as np

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

print (a.ndim)
```

output:2

NumPy Datatypes

The NumPy provides a higher range of numeric data types than that provided by the Python. A list of numeric data types is given in the following table.

SN	Data type	Description
1	bool_	It represents the boolean value indicating true or false. It is stored as a byte.
2	int_	It is the default type of integer. It is identical to long type in C that contains 64 bit or 32-bit integer.
3	intc	It is similar to the C integer (c int) as it represents 32 or 64-bit int.

4	intp	It represents the integers which are used for indexing.
5	int8	It is the 8-bit integer identical to a byte. The range of the value is -128 to 127.
6	int16	It is the 2-byte (16-bit) integer. The range is -32768 to 32767.
7	int32	It is the 4-byte (32-bit) integer. The range is -2147483648 to 2147483647.
8	int64	It is the 8-byte (64-bit) integer. The range is -9223372036854775808 to 9223372036854775807.
9	uint8	It is the 1-byte (8-bit) unsigned integer.
10	uint16	It is the 2-byte (16-bit) unsigned integer.
11	uint32	It is the 4-byte (32-bit) unsigned integer.
12	uint64	It is the 8 bytes (64-bit) unsigned integer.
13	float_	It is identical to float64.
14	float16	It is the half-precision float. 5 bits are reserved for the exponent. 10 bits are reserved for mantissa, and 1 bit is reserved for the sign.
15	float32	It is a single precision float. 8 bits are reserved for the exponent, 23 bits are reserved for mantissa, and 1 bit is reserved for the sign.
16	float64	It is the double precision float. 11 bits are reserved for the exponent, 52 bits are reserved for mantissa, 1 bit is used for the sign.
17	complex_	It is identical to complex128.
18	complex64	It is used to represent the complex number where real and imaginary part shares 32 bits each.
19	complex128	It is used to represent the complex number where real and imaginary part shares 64 bits each.

Operation between the array and scalar:

Arrays are important because arrays are enables you to express “Batch Operation” on data without writing the code using the loops. This process is called vectorization. Any arithmetic operations between equal-size arrays applies the operation element wise.

While performing these batch operations we should ensure that the arrays are of same size. We can also give unequal sizes for the arrays but the batch operations become complex.

Ex: import numpy as np

```
a=np.array([[1,2,3],[4,5,6]])
print (a+a)
print (a-a)
print (a*a)
print(a**a)
print(1/a)
```

output:

```
[[2 4 6]
 [8 10 12]]
```
(a+a)

```
[[0 0 0]
 [0 0 0]]
```
(a-a)

```
[[1  4  9]
 [16 25 36]]
```
(a*a)

```
[[    1     4    27]
 [256  3125 46656]]
```
(a**a)

```
[[1.         0.5        0.33333333]
 [0.25       0.2        0.16666667]]
```
(1/a)

Basic Indexing and Slicing:

NumPy array indexing is a rich topic, as there are many ways you may want to select a subset of your data or individual elements. One-dimensional arrays are simple.

- Indexing and slicing for two dimensional and multi-dimensional arrays are by applying memory diagram.
- For one dimensional the memory diagrams are simple and easy.

Ex: import numpy as np

```
    a=([0,1,2,3,4,5,6,7,8,9])
    b=a[4]
    c=a[3:6]
  print(b)
  print(c)
```

output: 4

[3,4,5]

Two dimensional array indexing:

```
import numpy as np
a=np.array([[1,2,3],[4,5,6],[7,8,9]])
b=a[0]
c=a[1][2]
d=a[1,2]
print(b)
print(c)
print(d)
```

output:

[1 2 3]

6

6

Three dimensional array indexing:

```
import numpy as np
a=np.array([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]]])
b=a[1]
print(b)
```

output:

[[7 8 9]

 [10 11 12]]

Boolean Indexing:

Up to now we indexed the ndarrays by using integers, it may be single (or) multi indexing. Now we are going to see indexing the ndarray by using Boolean.

Ex:

```
import numpy as np

names =np.array(['RAM','ANU','RAM','RAJ','ANU','RAJ','RAJ'])

print(names)

print(names== 'RAM')
```

output:

['RAM' 'ANU' 'RAM' 'RAJ' 'ANU' 'RAJ' 'RAJ']

[True False True False False False False]

- Now let us take the random values from random method 'np.random()' by using randn()

```
import numpy as np
data=np.random.randn(7,4)
print(data)
```

output:

```
[[0.30417723 0.41515405 -0.95040296 -1.26126195]
 [-0.70322128 -0.62610283 0.41852464 -0.05030524]
 [0.59810922 -1.62677969 -1.06821398 -0.08519538]
 [-0.34185538 0.99579905 0.34291575 -1.76710822]
 [-0.39285491 -0.27593837 -1.03252107 0.18550963]
 [0.75377994 0.29882073 -1.35004619 -0.16101322]
 [0.77611264 -0.66446863 0.98262041 -0.45447711]]
```

Now we are going to mapping the names array elements with random distribution data.

```
import numpy as np

names =np.array(['RAM','ANU','RAM','RAJ','ANU','RAJ','RAJ'])

data=np.random.randn(7,4)
```

```
print(names)

print(data)

c=data[names=='ANU']

print(c)
```

output:

```
['RAM' 'ANU' 'RAM' 'RAJ' 'ANU' 'RAJ' 'RAJ']

[[ 0.06635827  0.38594906  1.5808669   0.83691449]

 [-1.09703384  0.71375298 -0.676583   -1.93328441]

 [ 0.1393993  -0.61454576 -0.57447628 -0.33334039]

 [-1.14630241  1.69335773  0.07161279  1.11989231]

 [ 0.67996641  0.88623296 -2.24917515 -0.76058451]

 [ 1.34895545 -1.92956888  0.9647567  -0.7378757 ]

 [-1.19888797 -0.28596986 -1.66792688  1.25829226]]

[[-1.09703384  0.71375298 -0.676583   -1.93328441]

 [ 0.67996641  0.88623296 -2.24917515 -0.76058451]]
```

In Boolean indexing, the indexing will be done by integer also then

```
import numpy as np

names =np.array(['RAM','ANU','RAM','RAJ','ANU','RAJ','RAJ'])

data=np.random.randn(7,4)

print(names)

print(data)

c=data[names=='ANU',2:]

print(c)
```

output:

```
['RAM' 'ANU' 'RAM' 'RAJ' 'ANU' 'RAJ' 'RAJ']
```

[[0.69833327 -1.63990166 -0.33882464 -0.3435514]

[-1.31727695 0.82726501 -0.99366093 -1.82798608]

[-0.77404819 -0.74756308 1.48376161 0.31222039]

[-0.20824319 0.15961063 0.74663945 -0.66148184]

[0.00782858 0.99551292 -0.81444647 0.20370772]

[0.33594024 1.34310706 -0.17391126 1.21569313]

[-0.04567167 0.84650279 -1.03183314 1.24933909]]

[[-0.99366093 -1.82798608]

[-0.81444647 0.20370772]]

Fancy Indexing:

Fancy indexing is a term adopted by NumPy to describe indexing using integer arrays.

(or)

Fancy indexing is conceptually simple: it means passing an array of indices to access multiple array elements at once.

Ex:

```
import numpy as np

a=np.empty((8,4))

print(a)
```

output:

[[6.23042070e-307 4.67296746e-307 1.69121096e-306 6.23058707e-307]

[2.22526399e-307 6.23053614e-307 7.56592338e-307 1.60216183e-306]

[7.56602523e-307 3.56043054e-307 1.37961641e-306 2.22518251e-306]

[1.33511969e-306 6.23036978e-307 6.23053954e-307 9.34609790e-307]

[8.45593934e-307 9.34600963e-307 1.86921143e-306 6.23061763e-307]

[8.90104239e-307 6.89804132e-307 9.34605716e-307 1.37962456e-306]

[1.42418172e-306 8.06609645e-308 6.89810244e-307 1.22387550e-307]

[2.22522596e-306 8.34423917e-308 9.79107193e-307 9.79032499 e-307]]

```
import numpy as np
a=np.empty((8,4))
for i in range(8):
    a[i]=i
b=a[[3,1,7,4],[0,3,1,2]]
print(b)
```

output:

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

Here it prints the values from the data set a by treating first element of first array as arrow with first element of second array as column. In fancy indexing we can use negative indices for indexing.

```
import numpy as np
a=np.empty((8,4))
for i in range(8):
    a[i]=i
b=a[[-3,-1,-5]]
print(b)
```

output:

[[5. 5. 5. 5.]

[7. 7. 7. 7.]

[3. 3. 3. 3.]]

To prepare a data set of a specified range with required dimensions then we can use arrange().reshape()

```
import numpy as np

a=np.arange(32).reshape((8,4))

b=a[[0,2,5,4],[0,3,1,2]]

print(a)
```

output:

[[0 1 2 3]

[4 5 6 7]

[8 9 10 11]

[12 13 14 15]

[16 17 18 19]

[20 21 22 23]

[24 25 26 27]

[28 29 30 31]]

[0 11 21 18]

Data processing using arrays:

With the NumPy package, we can easily solve many kinds of data processing tasks without writing complex loops. It is very helpful for us to control our code as well as the performance of the program. In this part, we want to introduce some mathematical and statistical functions.

Expressing conditional logic as array operations:

Conditional operator (or) ternary operator is used to evaluate the expression based on condition. If condition is satisfy we will execute certain set of instructions otherwise other set of instruction. The numpy.where function is a vectorized version of the ternary expression x if condition else y.

Condition?: Expression 1|Expression 2

Ex:

```
import numpy as np

x=np.array([1.1,1.2,1.3,1.4,1.5])

y=np.array([2.1,2.2,2.3,2.4,2.5])

z=np.array(['True','False','True','False','True'])

r=[(x if z else y) for x,y,z in zip (z,x,y)]

print(r)
```

output:

```
[ 1.1  2.2  1.3  2.4  1.5]
```

Take a random (4,4) and replace 7 if it is greater than zero.

```
import numpy as np

a=np.random.randn(4,4)

print(a)

b=np.where(a>0,7,a)

print(b)
```

output:

```
[[-0.7206247   0.15346255 -0.34528202 -0.37227222]

 [ 0.59167644 -0.37805752 -0.53482493 -0.85060686]

 [-1.96227563  1.15500718  1.26861103  0.28010627]

 [ 1.87827211 -0.97443354  0.29180201  0.85444473]]

[[-0.7206247   7.         -0.34528202 -0.37227222]

 [ 7.         -0.37805752 -0.53482493 -0.85060686]

 [-1.96227563  7.         7.         7.        ]

 [ 7.         -0.97443354  7.         7.        ]]
```

Methods for Boolean arrays:

all(): This method returns “True” if all the elements of Boolean array is true ,otherwise it will return False.

In the following example given, it checks the arrays of elements are true then it returns True if all the elements of the array are true, otherwise False.

```
import numpy as np

a=np.array([True,False,True,True,False,False])

result=np.all(a)

print(result)
```

output:

False

any(): This method returns “True ” if anyone of the elements in the Boolean array is true ,otherwise it will return “False”.

In the following example ,it checks the arrays of elements are true then it returns True if any the elements of the array are true, otherwise False.

```
import numpy as np

a=np.array([True,False,True,True,False,False])

result=np.any(a)

print(result)
```

output:

True

Sorting of ndarrays:

In python, we can sort the elements of the given array in a sorted order. NumPy arrays can be sorted in-place using the sort method.

```
import numpy as np

a=np.array([ 0.6903, 0.4678, 0.0968, -0.1349, 0.9879, 0.0185, -1.3147,

 -0.5425])

result=a.sort()

print(a)
```

output:

[-1.3147 -0.5425 -0.1349 0.0185 0.0968 0.4678 0.6903 0.9879]

```
import numpy as np
a=np.random.randn(3,3)
print(a)
a.sort()
print(a)
```

output:

```
[[-0.95206174 -1.62759082  0.11736583]
 [ 0.34869056  0.22457328  1.2859413 ]
 [-0.91953969 -0.76986525  0.95541918]]
[[-1.62759082 -0.95206174  0.11736583]
 [ 0.22457328  0.34869056  1.2859413 ]
 [-0.91953969 -0.76986525  0.95541918]]
```

```
import numpy as np
a=np.random.randn(3,3)
print(a)
a.sort(1)
print(a)
```

output:

```
[[ 0.89501653 -0.38986388  0.43907009]
 [ 2.27251558  0.63101024  1.02060647]
 [ 0.26404843  0.58988295 -0.74397633]]
[[-0.38986388  0.43907009  0.89501653]
 [ 0.63101024  1.02060647  2.27251558]
```

[-0.74397633 0.26404843 0.58988295]]

Unique and set operations:

Unique:

This numpy set operation helps us find unique values from the set of array elements in Python. The numpy.unique() function skips all the duplicate values and represents only the unique elements from the Array.

Syntax: `np.unique (array)`

`Ex:`

```
import numpy as np

a = np.array([30,60,90,30,100])

data = np.unique(a)

print(data)
```

output:

[30 60 90 100]

Set union operation on NumPy Array:

NumPy offers us with universal union1d() function that performs UNION operation on both the arrays.

That is, it clubs the values from both the arrays and represents them. This process completely neglects the duplicate values and includes only a single occurrence of the duplicate element in the UNION set of arrays.

Syntax: `np.union1d(array, array)`

Ex:

```
import numpy as np
a1 = np.array([30,60,90,30,100])
a2 = np.array([1,2,3,60,30])
 data = np.union1d(a1,a2)
 print(data)
```

output:

[1 2 3 30 60 90 100]

Set Intersection operation on NumPy array:

The intersect1d() function enables us to perform INTERSECTION operation on the arrays. That is, it selects and represents the common elements from both the arrays.

Syntax: `np.intersect1d(array,array,assume_unique)`

assume_unique: If set to TRUE, it includes the duplicate values for intersection operation. Setting it to FALSE, would result in the neglection of duplicate values for intersection operation.

Ex:

```
import numpy as np

arr1 = np.array([30,60,90,30,100])

arr2 = np.array([1,2,3,60,30])

data = np.intersect1d(arr1, arr2, assume_unique=True)

print(data)
```

output: [30 30 60]

Finding uncommon values with NumPy Array:

With setdiff1d() function, we can find and represent all the elements from the 1st array that are not present in the 2nd array according to the parameters passed to the function.

Ex:

```
import numpy as np
a1 = np.array([30,60,90,30,100])
a2 = np.array([1,2,3,60,30])
data = np.setdiff1d(a1, r2, assume_unique=True)
print(data)
```

output:

[90 100]

Symmetric Differences:

With setxor1d() function, we can calculate the symmetric differences between the array elements. That is, it selects and represents all the elements that are not common in both the arrays. Thus, it omits all the common values from the arrays and represents the distinct values with respect to both the arrays.

Ex:

```
import numpy as np
a1 = np.array([30,60,90,30,100])
a2 = np.array([1,2,3,60,30])
 data = np.setxor1d(a1, a2, assume_unique=True)
 print(data)
```

output:

```
[1   2   3   90   100]
```