**Numpy**

* NumPy is a Python library used for working with arrays.
* NumPy arrays are stored at one continuous place in memory unlike lists, so processes can access and manipulate them very efficiently.
* The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.
* NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an **[ndarray](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html" \l "numpy.ndarray" \o "numpy.ndarray)** will create a new array and delete the original.
* The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory
* **Array Creation:**
* **import numpy as np**
* **np.array()🡺 Create an array from lists or tuples.**
* **np.zeros(shape,dtype=float)** **🡺Create an array of zeros with shape and type of input. #shape is int or tuple & dtype is optional.**
* **np.ones(shape, dtype=float)** **🡺 Create an array of ones with shape and type of input.**
* **np.empty(shape, dtype=float)** **🡺Create an empty array(uninitialized).**
* **np.full(shape, fill\_value)** **🡺 Create an array filled with a constant value(fill value).**
* **np.arange(start, stop, step)** **🡺 Create an array with evenly spaced values within a given range. #stop not included & default start is zero**
* **np.linspace(start, stop, num,endpoint)** **🡺 Create an array of evenly spaced numbers.#stop is included if endpoint=True otherwise it is excluded(endpoint=False)**
* **np.eye(N)** **🡺 Return a 2-D array with ones on the diagonal and zeros(identity matrix) #N:number of rows in the output**
* **np.random.rand(shape)** **🡺Generate an array of random values between 0 and 1.**
* **np.random.randint(low, high, size)** **🡺 Create an array of random integers from low(included)to high(excluded). #size is the shape of the output.**
* **Basic functions:**
* **array.ndim🡺 to Check how many dimensions the arrays have #0D,1D,2D,3D**
* **ndmin=5 argument🡺 to give the array specific number of dimensions.**
* **array.shape🡺 Get the shape of the array.**
* **array.size🡺 Get the total number of elements.**
* **array.type🡺 Get the data type of the array.**
* **array.dtype🡺 Get the data type of the array elements.**
* **array.itemsize🡺 Get the size of each element in bytes.**
* **array.nbytes🡺 Get the total number of bytes consumed by the array.**
* **Copying Arrays:**
* **np.copy(array)** **🡺 Return a copy of the array. #The copy NOT be affected by the changes made to the original array.**
* **array.view()🡺 Create a new view of the array with the same data. #he view SHOULD be affected by the changes made to the original array.**
* **Indexing, Slicing, and Iterating:**
* **array[index]** **🡺 Access a specific element of array.**
* **array[start:stop:step]** **🡺Slice a portion of the array**
* **array[:, i]** **🡺 Select all rows from a specific column**
* **array[i, :]** **🡺 Select all columns from a specific row**
* **np.where(condition,x,y)** **🡺 Return elements chosen from x or y depending on condition. #x if true, y if false.**
* **np.take(array, indices)** **🡺get elements from the array by indices.**
* **np.put(array, indices, values)** **🡺Place values in specific indices.**
* **Statistical Operations:**
* **np.sum(array)** **🡺 Sum of array elements.**
* **np.mean(array)** **🡺 Mean of array elements.**
* **np.median(array)** **🡺Median of array elements.**
* **np.std(array)** **🡺Standard deviation.**
* **np.var(array)** **🡺 Variance of array elements.**
* **np.min(array)** **🡺get Minimum value**
* **np.max(array)** **🡺get maximum value.**
* **np.cumsum(array)** **🡺Cumulative sum of elements.**
* **np.cumprod(array)** **🡺Cumulative product of elements.**
* **Math Functions:**
* **np.add(array1, array2)** **🡺 Addition of arrays.**
* **np.subtract(array1, array2)** **🡺Subtraction of arrays.**
* **np.multiply(array1, array2)** **🡺Multiplication of arrays.**
* **np.divide(array1, array2)** **🡺 Division of arrays.**
* **np.mod(array1, array2)** **🡺 Modulus of arrays.**
* **np.power(array1, array2)** **🡺 Exponentiation.**
* **np.sqrt(array)** **🡺Square root of each element.**
* **np.sin(array)** **🡺sin of array elements.**
* **np.cos(array)** **🡺 cos of array elements.**
* **np.tan(array)** **🡺tan of array elements.**
* **np.exp(array)** **🡺Exponentiation of each element.**
* **np.log(array)** **🡺Natural logarithm.**
* **np.dot(array1, array2)** **🡺Dot product of two arrays.**
* **np.abs(array) 🡺 Absolute values of elements**
* **np.greater(arr1,arr2) 🡺comparison of two arrays**
* **np.less(arr1,arr2) 🡺comparison of two arrays**
* **Array Reshaping and Transposition:**
* **array.reshape(new\_shape)** **🡺Change the shape of an array.**
* **array.ravel()🡺 Flatten the array to 1D.**
* **array.T🡺Get the transpose of the array.**
* **array.transpose(axes): Transpose the array**
* **np.expand\_dims(array, axis)** **🡺Expand the shape of an array by Adding an axis to an array. #axis=0 expand rows , =1 expand columns**
* **Array Concatenation and Splitting:**
* **np.concatenate(arrays, axis)** **🡺 Concatenate multiple arrays along an axis.**
* **np.vstack(arrays)** **🡺 Stack arrays vertically (row-wise),** **The arrays being stacked must have the same number of columns**
* **np.hstack(arrays)** **🡺Stack arrays horizontally (column-wise).**
* **np.dstack(arrays)** **🡺Stack arrays along the depth (third) axis.**
* **np.split(array, indices, axis)** **🡺Split an array into multiple sub-arrays.**
* **np.array\_split(array, indices, axis)** **🡺 Split array into sub-arrays of specific sizes.**
* **np.stack(arrays, axis)** **🡺 Stack arrays along a new axis.**

--------------------------------------------------------------------------------------------------------------------------------------

**Pandas DataFrame**

* **Pandas DataFrame** is two-dimensional data structure.
* Data is aligned in a tabular fashion in rows and columns.
* Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary
* **import** **pandas** **as** **pd**
* **creating a Dataframe:**
* **DataFrame can be created using a single list or a list of lists.**
* **create DataFrame from dict of narray/list, all the narray must be of same length.**
* **pd.DataFrame(data) 🡺 data like dictionaries, lists, or numpy arrays.**
* **Viewing Data:**
* **df.head(n) 🡺 View the first n rows of the DataFrame (default is 5).**
* **df.tail(n)** **🡺 View the last n rows of the DataFrame (default is 5).**
* **df.info()🡺Get a summary of the DataFrame, including column types and non-null values.**
* **df.describe()🡺Generate descriptive statistics of numeric columns.**
* **df.shape 🡺Returns a tuple representing the number of rows and columns.**
* **df.columns🡺List the column labels of the DataFrame.**
* **df.index🡺Get the row labels (indices) of the DataFrame.**

* **Selecting Data:**
* **df['column\_name'] 🡺select Single column**
* **df[['col1', 'col2']] 🡺select Multiple columns**
* **df.iloc[0]** **🡺select First row by index**
* **df.loc[0] 🡺 select First row by index label**
* **df[condition]** **🡺Conditional selection**
* **Modifying Data:**
* **df['new\_col'] = [value1, value2, value3]** **🡺adding a new column**
* **df.rename(columns={'old\_name': 'new\_name'}, inplace=True)** **🡺Renaming columns**
* **df.drop(['column\_name'], axis=1, inplace=True)** **🡺Drop a column**
* **df.drop([0], axis=0, inplace=True) 🡺Drop a row by index**
* **df['column\_name'].replace(old\_value, new\_value)** **🡺Replace values**
* **Sorting Data:**
* **df.sort\_values(by='column\_name', ascending=True)** **🡺sorting by a column**
* **df.sort\_index(ascending=True)** **🡺Sorting by index**
* **Handling Missing Data:**
* **df.isnull().sum()** **🡺Check missing values per column**
* **df['column\_name'].fillna(value, inplace=True)** **🡺Filling missing values**
* **df.dropna(subset=['column\_name'], inplace=True)** **🡺**

**Drop Rows/Columns of datasets with Null values**

* **Aggregating Data:**
* **df.groupby('column\_name').agg({'column\_name': 'sum'})🡺Grouping and aggregating data**
* **df['column\_name'].sum()** **🡺suming values of a column**
* **df['column\_name'].mean()🡺get mean of a column**
* **df['column\_name'].min()🡺get minimum value of a column**
* **df['column\_name'].max()🡺get maximum of a column**
* **Iterating over Rows:**
* **df.iterrows()**
* **df.itertuples()**
* **df.iteritems()**
* **iterating over Columns:**
* **In order to iterate over columns, we need to create a list of dataframe columns and then iterating through that list to pull out the dataframe columns.**
* **Merging** &**Joining DataFrames**:
* **df1.join(df2, how='left')** **🡺 combining two DataFrames with the same index.**
* **pd.merge(df1, df2, on='common\_column', how='left')** **🡺** **merge DataFrames based on one or more columns.**

-----------------------------------------------------------------------------------------------------------------------------------------

**Dictionary**

* Dictionaries are used to store data values in key:value pairs.
* A dictionary is a collection which is ordered, changeable and do not allow duplicates.
* Dictionaries are written with curly brackets, and have keys and values:
* You can access the items of a dictionary by referring to its key name, inside square brackets.
* dic.get(“key”)to get the value of the key.
* dic.keys() to return a list of the keys.
* The list of the keys is a view of the dictionary, meaning that any changes done to the dictionary will be reflected in the keys list.
* dic.values() return a list of all the values in the dictionary.
* dic.items()return each item in a dictionary, as tuples in a list.
* dic[“key”]=”value” to add item in dictionary.
* To determine if a specified key is present in a dictionary use the in keyword.
* can change the value of a specific item by referring to its key name or using update()method.
* dic.Update({"key": value})  update the dictionary with the items from the given argument.
* dic.pop("key") remove item with specified key name.
* dic.popitem() remove the last inserted item.
* The **del** keyword removes item with specified key name or delete the dictionary completely. del dic[“key”]
* dic.clear()
* When looping through a dictionary, the return value are the keys of the dictionary.
* dic2=dic1.copy() make a copy of dictionary or using dict() method.
* A dictionary can contain dictionaries, this is called nested dictionaries. myfamily = {  
     "child1" : {  
       "name" : "Emil",  
       "year" : 2004  
    },  
    "child2" : {  
       "name" : "Tobias",  
       "year" : 2007  
     },  
     "child3" : {  
       "name" : "Linus",  
       "year" : 2011  
     }  
   }
* To access items from a nested dictionary, you use the name of the dictionaries, starting with the **outer** dictionary dic["outerkey"]["innerkey"]).
* thisdict = dict.fromkeys(x,y) method is used to create a new dictionary and assign a standard value to all of its keys x = ('key1', 'key2', 'key3')  
  y = 0  
  thisdict = dict.fromkeys(x, y)
* dic.setdefault("key", "value") returns the value of the item with the specified key,If the key does not exist, insert the key, with the specified value.