NUMPY

numpy as np

np.__version__

type(array_variable)

np.array([_,_,_,_])

array_variable.ndim

array_variable.shape

array_variable(operator)numeric

variable = np.random.default_rng()
variable.integers(low = {numeric},high = {numeric},size=
(number_of_matrix,rows,coloumns))
new_variable = np.random.default_rng(seed = 1)

for floating pointers:-
np.random.uniform(numeric},high = {numeric},size=
(number_of_matrix,rows,coloumns)

for shuffling:-
varibale.shuffle(array_variable)

Slicing
variable_name[start:end:step]
for slicing coloumn section :-
variable_name[row_section,coloumn_section]
if want to display all elements in row and edit in
coloumn make sure to place a clon in row side
ex:- s[:,1:6:2]

Filtering
array_variable = array_variable[condition with
array_variable]
use & , | , ^ (similar to C language)
np.where(condition , array_variable ,
replicateed_value_if_condition is false)

Vectorized math functions
np.sqrt(array_variable)
np.round(array_variable)
np.floor(array_variable)
np.ceil(array_variable)
np.pi

Broadcasting
used to virtually expand
dimensions
ex array_variable *
array_variable

Aggregate Functions
np.sum(array_variable)
np.mean(array_variable)
np.std(array_variable)
np.var(array_variable)
np.min(array_variable)
np.max(array_variable)
np.argmin(array_variable)
np.argmax(array_variable)
np.sum(array_variable,axis=1) #r
np.sum(array_variable,axis=0) #c

any condition written using array_
variable inside print or into a variable
gives/ results the elements which
satisfy the condition
ex:- scores[scores < 60] = 0
print(scores[scores >= 90])

array_variable (arthimetic_operator)
array_variable

Pandas

pandas as pd

df = np.read_csv("r or a")
df = np.read_json("r or a")

we can add a parameter index_col = "column_name"
this helps to set an column as indexing

df.to_string()
to_string is used to display entire data , if not we only get first and last 5 lines of data

SELECTION BY columnS

df["column_name , column_name , ..."].to_string()

SELECTION BY ROWS

only one row_name :-
df.loc["row_name"]

For multiple rows and column changes :-
df.loc([row_selection , column_selection])

we can also use position numbers but through .iloc
df.iloc([row_seelction.column_selection])

Series :- it is a 1-DImentional labeled array can hold any datatype
Think of a single column in a spreadsheet

pd.Series(data_variable)
we can add another parameter index="index_name1,index_name2,..."

Aggregate functions :- reduces a set of values into a single summary value
df.mean(numeric_only = True)
df.sum(numeric_only = True)
df.min(numeric_only = True)
df.max(numeric_only = True)
df.count()

Single column:-
df["column_name"].mean()
df["column_name"].sum()
df["column_name"].min()
df["column_name"].max()
df["column_name"].count()

using groupby()
variable = df.groupby("column_name")
variable["other_column_name"].mean()
variable["other_column_name"].sum()
variable["other_column_name"].min()
variable["other_column_name"].max()

Flitering :- keeping the rows that matching the c
df[condition]
for multiple conditions should use & , | ,

Data Frames

variable_data = {
"column_1": [value1,value2,value3,..]
"column_2": [value1,value2,value3,..]
}
df = pd.dataFrame(variable_data,inddex = ["row1","r

Adding a new column :-
df["new_column"] = [value1,value2,value3

Adding a new row :-
new_row = pd.DataFrame(["variable1":"value1","variable2"
[row1,row2,row3])
df = pd.concat(df,new_row)
Note :- the way you place parameters in concat() the data fran

Data Cleaning :- The process of fixing / removing incomplete , irre

1. Droping irrelavent columns
df = df.drop(columns = [column1,column2,..

2. removing rows with NaN values wrt coloun
df = df.dropna(subset=[column1,column2,...

3. Filling the NaN values
df = df.fillna({"coloumn_name": value})

4. Fix inconsistant values
df["coloumn_name"] = df["coloumn_name"].repl
"value1": "change1",
"value2": "change2"
....

condition

^

row2","row3"])

3,..]

': "value2",...],index =

me changes accordingly

elavent , incorrect data .

.])

nns
.])

lace({

})