Basic Python

01-Our First Program

In [ ]:

```
# Write print and then () and inside bracket add "" and write any thing you want to pr
print("Hello World")
print(2+3)
print("Python with Ammar")
```

```
Hello World
5
Python with Ammar
```

02-Operators

In [ ]:

```
# Addition
print(2+3)
# Subtraction
print(3-2)
# Multiplication
print(48*3)
#Divivion for getting a floating no. like 3.25
print(22/3) #floating numbers
# Division for getting a whole no
print(11//3) #for whole number
# Getting a power
print(2**3)
# Getting a percentage
print(34%2)
# Adding simultanioulу all funtion according to PEMDAS
print(2**3/2*3/3+6-4+2)

#PEMDAS
#Parenthesis Exponenets Multiply Addition Substraction
#left to right sequencefor M D & A S*
```

```
5
1
144
7.333333333333333
3
8
0
8.0
```

03-Strings

In [ ]:

```
# Anything we write inside the "" is our string
print("Hello World")
print("Python with Ammar")
print('test for single quote')
print("test for double quote")
print('''test for trriple quotes''')
print("Whats's Up")
print("  what's up    ?")
print("srtring_clear")
```

```
Hello World
Python with Ammar
test for single quote
test for double quote
test for trriple quotes
Whats's Up
  what's up    ?
srtring_clear
```

04-Comments

In [ ]:

```
# Adding anything to comment by pressing (Ctrl+/)
print("How are you?") #press these to comment out (Ctrl+/)
print (" We are learning python with ammar") #print a string
print(2+3) #print operators function withnumbers
```

```
How are you?
 We are learning python with ammar
5
```

05-Variable

In [ ]:

```
#Variables: objects containing specific values
x=5 #Numeric or integer variable
print (x)
y= "we are learning Python with ammar" #string variable
print(y)
x= x+15
print(x) #answer will be 20 beacause lines are updating from upward to downward contir

#types/class of variables
type(x)
print(type(x))
print(type(y))

#print_type_class

#rules to assign a variable
#1 . Variable should contains letter , numbers or underscores
#2 . Do not start with numbers Like 2y instead of y only
#3 . Spaces are not allowed in variable name
#4 . Do not use keywords used in functions like break ,  mean , media, test, etc
#5 . Short and Descriptive
#6 . Case sensitivity (Lowercase , upper case letters , lower case letters should be u

fruit_basket= 10
fruit_basket= "Mangoes"
print(type (fruit_basket))
#del fruit_basket
print(fruit_basket)
```

```
5
we are learning Python with ammar
20
<class 'int'>
<class 'str'>
<class 'str'>
Mangoes
```

06-Input Variables

In [ ]:

```python
# greetings= "Assalam-u-Alikum "
# asking=",kia hal hain?"
# print(greetings,name,asking)

#3rd Stage input function
name=input("what is your name? ")
age=input("How old are you? ")
greetings="Hello"

print(greetings,name,",you are still young bro ")

#input_ammar_ You are still young
```

```
Hello Arslan ,you are still young bro
```

07-Conditional Logics

In [ ]:

```python
#logical operators are either "true or false or yes or no or 0 or 1"
#equal to                   ==
#not equal to               !=
#less than                  <
#greater than               >
#less then and equal to     <=
#greater than and wqualto   >=

# question.. is 4 equal to 4
print(4==4)
print(4!=4)
print(4>3)
print(5<4)
print(3<=5)
print(3>=5)
```

```
True
False
True
False
True
False
```

In [ ]:

```python
# #application of logical operators
hammad_age=4
age_at_school=5
print(hammad_age==age_at_school)
```

```
False
```

In [ ]:

```python
#input function and logical operator
age_at_school= 5 #variable
```

```
hammad_age=input("What is the age of Hammad ?") #input function
hammad_age=int(hammad_age) #changing type of variavble
print(type(hammad_age))
print(hammad_age==age_at_school) #logical operator

#convert input
```

```
<class 'int'>
False
```

08-Type Conversion

In [ ]:

```
x=10        #integer
y=10.2      #float
z="hello"   #string

print(type(x))
print(type(y))
print(type(z))

#implicit type conversion
x=x*y

print(x, type(x))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
102.0 <class 'float'>
```

In [ ]:

```
#explicit type conversion
age=input("what is your age? ")
print(type(float(age)))
age=int(age)
print(type(int(age)))
print(age,type(str(age)))
```

```
<class 'float'>
<class 'int'>
24 <class 'str'>
```

In [ ]:

```
name=input("what is your name ?")
print(name, type(str(name)))
```

```
Arslan <class 'str'>
```

09-if, else and elif

In [ ]:

```
required_age_at_school=4
hammad_age=1

#question: Can hammad goto school
if hammad_age==required_age_at_school:
    print("Congratulation!! Hammad can join the school")
elif hammad_age > required_age_at_school :
    print("Hammad should join higher school")
elif hammad_age<=2:
    print("You should take care of Hammad he is still a baby ")
else:
    print("Hammad Can not go to school ")

#i, elif, else statement clear .
```

```
You should take care of Hammad he is still a baby
```

10-Functions

In [ ]:

```
#1

#defining a functions
def print_codanics():
     print("We are learning with ammar")
     print("We are learning with ammar")
     print("We are learning with ammar")

print_codanics()
```

```
We are learning with ammar
We are learning with ammar
We are learning with ammar
```

In [ ]:

```
#2
def print_code():
    text= "we are learning python with ammar "
    print(text)
    print(text)
    print(text)

print_code()
```

```
we are learning python with ammar
we are learning python with ammar
we are learning python with ammar
```

In [ ]:

```
#3
def print_code(text):
    print(text)
    print(text)
    print(text)

print_code("We are learning python")
```

```
We are learning python
We are learning python
We are learning python
```

In [ ]:

```
#4
#defining a function with if elif and else statement

def school_calculator(age):
    if age==5:
        print("Hammad can join the school")
    elif age>5:
        print("Hammad should go to higher school")
    else:
        print("Hammad is still a baby")
```

In [ ]:

```
# school_calculator(5)
```

```
#defining a function of future
def future_age(age):
    new_age= age+20
    return new_age
    print(new_age)
    # print(new_age)
furture_age=future_age(3)
print(furture_age)
```

```
23
```

In [ ]:

```
#i understand functions really well

def repeat_arslan_4times():
    text= ("Arslan")
    print(text)
    print(text)
    print(text)
    print(text)

repeat_arslan_4times()
```

```
Arslan
Arslan
Arslan
Arslan
```

In [ ]:

```
name=input("What is your name? ")
age=int(input("What is your age ? "))
greetings= ("Hello")

def school_extrance_calculator(age):
    print(greetings,name)
    if age>=5 and age<9:
        print("You are welcome to school")
    elif age<5:
        print("You are not eligible")
    elif age>=10 and age<15:
        print("You should go to higher school")
    else:
        print("you should go to university")

school_extrance_calculator(age)
```

```
Hello arslan
You are welcome to school
```

11-Loops

while loops and for loops

In [ ]:

```
##while loops

x=0
while(x<=5):
    print (x)
    x=x+1
```

```
0
1
2
3
4
5
```

In [ ]:

```
#for loop
for x in range (4,11):
    print(x)
```

```
4
5
6
7
8
9
10
```

In [ ]:

```
#array
days= ["mon", "tue", "wed", "thurs", "fri", "sat", "sund"]
for d in days:
    if d=="fri": break #Stop the loop
    # if d=="fri": continue #skip that entity
    print(d)
```

```
mon
tue
wed
thurs
```

12-Import Library

In [ ]:

```
#if you want to print the value of pi
import math
print("The value of pi is ",math.pi)
print(type(math.pi))
```

```
The value of pi is  3.141592653589793
<class 'float'>
```

In [ ]:

```
import statistics
x= [150, 250,350,450]
print(statistics.mean(x))
#some important libraries
#numpy, pandas
```

```
300
```

13-TroubleShooting

In [ ]:

```
#print(We are learning) #syntax error
#print(25/0) #runtime error

name="ammar"
print("Hello", name)

#trouble shooting is easy
```

```
Hello ammar
```

14-Practice

In [ ]:

```
name= input("What is your name ? ")
print(name)
age= int(input("What is your age? "))
print(age)
print(type(age))

if age==24:
    print(name,"You are still young bro")
elif age<24:
    print(name, "you are still a baby")
elif age>24 and age<100:
    print(name ,"get marry please")
elif age>=100 and age<200:
    print(name, "You are suppose to dead")
else:
    print("Dead")
```

```
Arslan
50
<class 'int'>
Arslan get marry please
```

BMI Calculator

In [ ]:

```
#units of BMI is wiegh in kg / height in m and its square
name= input("What is your name ?  ")
greetings= ("Hello" ,name)
greetings
weight= float(input("what is your weight?  "))
height= float(input("and your height ?  "))


bmi= weight/height**2
bmi

print(name ,"your BMI IS", bmi)
```

```
arslan your BMI IS 0.0035083029837281567
```

Indexing

In [ ]:

```
#make a string
a= "Samosa Pakora"
a
```

```
'Samosa Pakora'
```

In [ ]:

```
#checking the value at index 0
a[0]
#counting will star from 0 to onwards in python
```

```
'S'
```

```
In [ ]: a[1]
```

```
'a'
```

```
In [ ]: a[2]
```

```
'm'
```

```
In [ ]: a[6]
        #it will print a space
```

```
' '
```

```
In [ ]: len(a)
        #it will show the number of index in our string
```

```
13
```

```
In [ ]: a[0:6]
        #the last no will be excludes like i ask for letters from 0 to 6 but it will print fro
```

```
'Samosa'
```

```
In [ ]: a[1:8]
        #here P is 7th character as we strat counting from 0
```

```
'amosa P'
```

```
In [ ]: a[0:13] #here if we count from 0 to 13 it will be total 14 characters here 13 no is ex
```

```
'Samosa Pakora'
```

```
In [ ]: a[-2]
        #here it starts from right side and will start from number (-1)
```

```
'r'
```

```
In [ ]: a[-1:-6]

        #here it will not print any thing
```

```
''
```

```
In [ ]: a[-6:-1]
        #here we see the writing sequence in string will remain same from right to left
        #also -1 no is "a" but it will not print here as last no is exclude
```

```
'Pakor'
```

```
In [ ]: a[-6:0]
```

```
''
```

```
In [ ]: a[-6:13]
```

```
'Pakora'
```

```
In [ ]: food= "birYani"
        food
```

```
'birYani'
```

String Methods

In [ ]:

```
food
```

```
'birYani'
```

In [ ]:

```
#Checking the length
len(food)
```

```
7
```

In [ ]:

```
# Capitalize
food.capitalize()
```

```
'Biryani'
```

In [ ]:

```
#Upper case letters
food.upper()
```

```
'BIRYANI'
```

In [ ]:

```
#lower case letters
food.lower()
```

```
'biryani'
```

In [ ]:

```
#replace
food.replace("b", "sh")
```

```
'shirYani'
```

In [ ]:

```
#counting a specific alphabet in a string
name = "baba_aammar with Dr aamar tufail"
name
```

```
'baba_aammar with Dr aamar tufail'
```

In [ ]:

```
name.count("a")
```

```
9
```

In [ ]:

```
name.count("D")
```

```
1
```

In [ ]:

```
#how to find a number of index in string
name = "baba_aammar with Dr aamar tufail"
name
```

```
'baba_aammar with Dr aamar tufail'
```

In [ ]:

```
name.find("t")
```

```
14
```

In [ ]:

```
# how to split a string
food = "i love samosa , pakora , raita, biryani and karahi"
food
```

```
'i love samosa , pakora , raita, biryani and karahi'
```

In [ ]:

```
food.split(",")
```

```
['i love samosa ', ' pakora ', ' raita', ' biryani and karahi']
```

Basic data Structure in Python

1-Tuple

2-List

3-Dictionaries

4-Set

Tuple

- Ordered collection of elements
- eclosed in () round braces/ paranthesis
- Different kind of elements can be stored (elements like int, float , string, boolean{true, false})
- Once elements are stored you can not change or replace them (Unmutatable)

In [ ]:
```
tup1 = (1,"python" , True , 2.5)
tup1
```

```
(1, 'python', True, 2.5)
```

In [ ]:
```
#type of a tuple
type(tup1)
```

```
tuple
```

-indexing in tuple

In [ ]:
```
tup1[1]
```

```
'python'
```

In [ ]:
```
tup1[0]
```

```
1
```

In [ ]:
```
tup1[0:6]
```

```
(1, 'python', True, 2.5)
```

In [ ]:
```
tup1[0:3] # last element is exclusive
```

```
(1, 'python', True)
```

In [ ]:
```
#length of tuple
len(tup1)
```

```
4
```

In [ ]:
```
tup2 = (2, "baba ammar", 3.5, False )
```

```
tup2
```

```
(2, 'baba ammar', 3.5, False)
```

In [ ]:

```
# concatinate ( TO add two or more tuple)
tup1+tup2
```

```
(1, 'python', True, 2.5, 2, 'baba ammar', 3.5, False)
```

In [ ]:

```
#concatinate + repeat
tup1*3 + tup2
```

```
(1,
 'python',
 True,
 2.5,
 1,
 'python',
 True,
 2.5,
 1,
 'python',
 True,
 2.5,
 2,
 'baba ammar',
 3.5,
 False)
```

In [ ]:

```
tup1*2 + tup2
```

```
(1, 'python', True, 2.5, 1, 'python', True, 2.5, 2, 'baba ammar', 3.5, False)
```

In [ ]:

```
tup3 = (20, 50, 60, 80, 96)
tup3
```

```
(20, 50, 60, 80, 96)
```

In [ ]:

```
max(tup3)
```

```
96
```

In [ ]:

```
min(tup3)
```

```
20
```

In [ ]:

```
tup3*2
```

```
(20, 50, 60, 80, 96, 20, 50, 60, 80, 96)
```

List

- ordered collection of elements
- enclosed in [] square barckets
- Mutateable, you can change the values

In [ ]:

```
list1 = [2, "baba ammar" , False]
list1
```

```
[2, 'baba ammar', False]
```

```
In [ ]: type(list1)
```

```
list
```

```
In [ ]: len(list1)
```

```
3
```

```
In [ ]: list1[2]
```

```
False
```

```
In [ ]: list2 = [3, 5, "Aammar", "Codanics", 478, 53.2, True]
        list2
```

```
[3, 5, 'Aammar', 'Codanics', 478, 53.2, True]
```

```
In [ ]: list1 + list2
```

```
[2, 'baba ammar', False, 3, 5, 'Aammar', 'Codanics', 478, 53.2, True]
```

```
In [ ]: list1*2
```

```
[2, 'baba ammar', False, 2, 'baba ammar', False]
```

```
In [ ]: list1
```

```
[2, 'baba ammar', False]
```

```
In [ ]: list1.reverse()
        list1
```

```
[False, 'baba ammar', 2]
```

```
In [ ]: list1.append("codanics youtube channel")
        list1
```

```
[False, 'baba ammar', 2, 'codanics youtube channel']
```

```
In [ ]: list1.count(False)
```

```
1
```

```
In [ ]: list3 = [20,30,40,50,60,52,562,488,2485]
        list3
```

```
[20, 30, 40, 50, 60, 52, 562, 488, 2485]
```

```
In [ ]: len(list3)
```

```
9
```

```
In [ ]: #sorting a List
        list3.sort()
        list3
```

```
[20, 30, 40, 50, 52, 60, 488, 562, 2485]
```

```
In [ ]: #repeat
        list3*3
```

```
[20,
 30,
 40,
 50,
 52,
 60,
 488,
 562,
 2485,
 20,
 30,
 40,
 50,
 52,
 60,
 488,
 562,
 2485,
 20,
 30,
 40,
 50,
 52,
 60,
 488,
 562,
 2485]
```

In [ ]:
```
list2+list3
```

```
[3,
 5,
 'Aammar',
 'Codanics',
 478,
 53.2,
 True,
 20,
 30,
 40,
 50,
 52,
 60,
 488,
 562,
 2485]
```

In [ ]:
```
lists= list1 +list2
lists
```

```
[False,
 'baba ammar',
 2,
 'codanics youtube channel',
 3,
 5,
 'Aammar',
 'Codanics',
 478,
 53.2,
 True]
```

3- Dictionaries

- AN unirdered collection of element
- Key and value
- Curly braces/ braces {}
- Mutateable , you can change the value

In [ ]:

```
#Food and thier prices
food1= {"Samosa" : 30, "Pakora" : 100, "Raita" : 20, "Salad" : 50, "Chicken Rolls": 3
food1
```

```
{'Samosa': 30, 'Pakora': 100, 'Raita': 20, 'Salad': 50, 'Chicken Rolls': 30}
```

In [ ]:

```
type(food1)
```

```
dict
```

In [ ]:

```
#extract data
keys= food1.keys()
keys
```

```
dict_keys(['Samosa', 'Pakora', 'Raita', 'Salad', 'Chicken Rolls'])
```

In [ ]:

```
values = food1.values()
values
```

```
dict_values([30, 100, 20, 50, 30])
```

In [ ]:

```
#adding new element
food1["Tikki"]=10
food1
```

```
{'Samosa': 30,
 'Pakora': 100,
 'Raita': 20,
 'Salad': 50,
 'Chicken Rolls': 30,
 'Tikki': 10}
```

In [ ]:

```
#updating a values
food1["Tikki"]= 15
food1
```

```
{'Samosa': 30,
 'Pakora': 100,
 'Raita': 20,
 'Salad': 50,
 'Chicken Rolls': 30,
 'Tikki': 15}
```

In [ ]:

```
food2 = {"Dates": 50, "Chocolates":200, "Sawayyan":1000}
food2
```

```
{'Dates': 50, 'Chocolates': 200, 'Sawayyan': 1000}
```

In [ ]:

```
#Concatinate
food1.update(food2)
food1
```

```
{'Samosa': 30,
 'Pakora': 100,
 'Raita': 20,
 'Salad': 50,
 'Chicken Rolls': 30,
 'Tikki': 15,
 'Dates': 50,
 'Chocolates': 200,
 'Sawayyan': 1000}
```

4-Sets

- An unordered and un-indexed
- Curly braces {} are used
- no duplicates allowed

In [ ]:

```
s1= {1, 2, 2.2, 5, "Codanics", "Faisalabad", True}
s1
# here we see that boolean operator dosent print in sets
```

```
{1, 2, 2.2, 5, 'Codanics', 'Faisalabad'}
```

In [ ]:

```
s1.add("codanics")
s1
```

```
{1, 2, 2.2, 5, 'Codanics', 'Faisalabad', 'codanics'}
```

In [ ]:

```
s1.add("Faisalabad")
s1
```

```
{1, 2, 2.2, 5, 'Codanics', 'Faisalabad', 'codanics'}
```

In [ ]:

```
s1.remove("codanics")
s1
```

```
{1, 2, 2.2, 5, 'Codanics', 'Faisalabad'}
```

Numpy

1-D Array

In [ ]:

```
import numpy as np
```

In [ ]:

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

```
array([1, 2, 3, 4, 5])
```

In [ ]:

```
type(a)
```

```
numpy.ndarray
```

In [ ]:

```
len(a)
```

```
5
```

In [ ]:

```
# creating a single axis array of number zero
c= np.zeros(2)
c
```

```
array([0., 0.])
```

In [ ]:

```
# creating a single axis array of number One
d= np.ones(10)
d
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

In [ ]:

```
e=np.empty(3)
e
```

```
array([1.29822923e-311, 0.00000000e+000, 2.11382017e-307])
```

In [ ]:

```
# with the specific range of elements
g= np.arange(5,15) # as we arleady know the last no. is exclusive
g
```

```
array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [ ]:

```
# with the range of elements with a speciefied gap
h=np.arange(2,20,2) # goin from 2 to 20 with a specified gap of 2 and last no. is exlc
h
```

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18])
```

In [ ]:

```
# and if we want 20 no. also in last arange
h=np.arange(2,21,2)
h
```

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [ ]:

```
# Linearly spaced arrays
i= np.linspace(0,15 , num= 5) # GOing 0 to 15 in just 5 numbers in a way that the dist
i
```

```
array([ 0.  ,  3.75,  7.5 , 11.25, 15.  ])
```

In [ ]:

```
j= np.ones(5, dtype=np.float64)
j
```

```
array([1., 1., 1., 1., 1.])
```

2-D Array

In [ ]:

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

```
array([[2, 2, 2, 2],
       [3, 3, 3, 3]])
```

2- axis

In b

- First axis has a length = 2
- Second axis has lenght = 4

In [ ]:

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

```
array([[1, 1, 1, 1],
       [2, 2, 2, 2]])
```

In [ ]:

```
k=np.zeros((3,4))
k
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [ ]:

```
l=np.zeros((5,6))
l
```

```
array([[0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0.]])
```

In [ ]:

```
m=np.ones((2,4))
m
```

```
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

In [ ]:

```
d= np.array([[2,2,2],[2,2,2],[6,5,4]])
d
```

```
array([[2, 2, 2],
       [2, 2, 2],
       [6, 5, 4]])
```

In [ ]:

```
f= np.array ([[4,5,6,4],[8,6,4,5],[8,6,4,2],[9,6,3,2]])
f
```

```
array([[4, 5, 6, 4],
       [8, 6, 4, 5],
       [8, 6, 4, 2],
       [9, 6, 3, 2]])
```

In [ ]:

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

```
array([[1, 2, 3],
       [2, 3, 4],
       [4, 5, 6]])
```

3-D Array

In [ ]:

```
# TensorFlow is a library use for 3 Dimensional things
# TensorFlow is also a free and open source software libraryfor machine learning and a
```

In [ ]:

```
#making and reshaping a 3D array
c= np.arange(24) .reshape (2,3,4) # First axis has  length = 2,,Second Axis has length
c
```

```
array([[[ 0,  1,  2,  3],
        [ 4,  5,  6,  7],
        [ 8,  9, 10, 11]],

       [[12, 13, 14, 15],
        [16, 17, 18, 19],
        [20, 21, 22, 23]]])
```

In [ ]:

```
d= np.zeros((2,3,3))
d
```

```
array([[[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]])
```

In [ ]:

```
f= np.ones((3,4,5) , dtype= np.int64)
f
```

```
array([[[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],

       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]],

       [[1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1],
        [1, 1, 1, 1, 1]]], dtype=int64)
```

In [ ]:

```
f= np.array ([[[4,5,6,4],[8,6,4,5],[8,6,4,2],[9,6,3,2]],[[4,5,6,4],[8,6,4,5],[8,6,4,2]
f
```

```
array([[[4, 5, 6, 4],
        [8, 6, 4, 5],
        [8, 6, 4, 2],
        [9, 6, 3, 2]],

       [[4, 5, 6, 4],
        [8, 6, 4, 5],
        [8, 6, 4, 2],
        [9, 6, 3, 2]],

       [[4, 5, 6, 4],
        [8, 6, 4, 5],
        [8, 6, 4, 2],
        [9, 6, 3, 2]]])
```

In [ ]:

```
z= np.array ([[[1,2,3],[4,5,6]],[[7,8,9],[10,11,12]],[[13,14,15],[16,17,18]]])
z
```

```
array([[[ 1,  2,  3],
        [ 4,  5,  6]],

       [[ 7,  8,  9],
        [10, 11, 12]],

       [[13, 14, 15],
        [16, 17, 18]]])
```

Numpy Practice Session

In [ ]:

```
#importing numpy library
import numpy as np
```

creating an array using numpy

In [ ]:

```
import numpy as np
food= np.array(["pakora" , "samosa" , "raita"])
food
```

```
array(['pakora', 'samosa', 'raita'], dtype='<U6')
```

In [ ]:

```
price = np.array([5,5,5])
price
```

```
array([5, 5, 5])
```

In [ ]:

```
#Checking type of array
type(price)
```

```
numpy.ndarray
```

In [ ]:

```
type(food)
```

```
numpy.ndarray
```

In [ ]:

```
#length of array
len(food)
```

```
3
```

In [ ]:

```
#indexing
price[2]
```

```
5
```

In [ ]:

```
price[0:]
```

```
array([5, 5, 5])
```

In [ ]:

```
#index no to find the index in an array
food[1]
```

```
'samosa'
```

In [ ]:

```
price.mean()
```

```
5.0
```

In [ ]:

```
# zeros method
a= np.zeros(6)
a
```

```
array([0., 0., 0., 0., 0., 0.])
```

In [ ]:

```
# ones method
b= np.ones(5)
b
```

```
array([1., 1., 1., 1., 1.])
```

In [ ]:

```
c= np.empty(5)
c
```

```
array([1., 1., 1., 1., 1.])
```

In [ ]:

```
# Making a Range
a= np.arange(10)
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [ ]:

```
# Specified range
a= np.arange(2,21)
a
```

```
array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
       19, 20])
```

In [ ]:

```
# specific arang with specific distance
a= np.arange (2,20,3)
a
```

```
array([ 2,  5,  8, 11, 14, 17])
```

In [ ]:

```
#table of 5
a= np.arange (5,55,5)
a
```

```
array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])
```

In [ ]:

```
# with line space
a= np.linspace (0,10, num = 6 ,dtype= np.int64 )
```

```
a
```

```
array([ 0,  2,  4,  6,  8, 10], dtype=int64)
```

In [ ]:

```
b= np.linspace(1,100, num = 40)
b
```

```
array([  1.        ,   3.53846154,   6.07692308,   8.61538462,
        11.15384615,  13.69230769,  16.23076923,  18.76923077,
        21.30769231,  23.84615385,  26.38461538,  28.92307692,
        31.46153846,  34.        ,  36.53846154,  39.07692308,
        41.61538462,  44.15384615,  46.69230769,  49.23076923,
        51.76923077,  54.30769231,  56.84615385,  59.38461538,
        61.92307692,  64.46153846,  67.        ,  69.53846154,
        72.07692308,  74.61538462,  77.15384615,  79.69230769,
        82.23076923,  84.76923077,  87.30769231,  89.84615385,
        92.38461538,  94.92307692,  97.46153846, 100.        ])
```

In [ ]:

```
# specifing the data type
a= np.ones(5, dtype= np.int8)
a
```

```
array([1, 1, 1, 1, 1], dtype=int8)
```

In [ ]:

```
a= np.ones(50, dtype= np.float64)
a
```

```
array([1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.,
       1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1.])
```

Array functions

In [ ]:

```
a= np.array([ 10,12,15,2,4,6,18,100,18,16,10.3,0.5])
a
```

```
array([ 10. ,  12. ,  15. ,   2. ,   4. ,   6. ,  18. , 100. ,  18. ,
        16. ,  10.3,   0.5])
```

In [ ]:

```
#sorting an array
a.sort()
a
```

```
array([  0.5,   2. ,   4. ,   6. ,  10. ,  10.3,  12. ,  15. ,  16. ,
        18. ,  18. , 100. ])
```

In [ ]:

```
b= np.array([10.5,5,15.6,8,0.5,10.5,100.9,15,16,59])
b
```

```
array([ 10.5,   5. ,  15.6,   8. ,   0.5,  10.5, 100.9,  15. ,  16. ,
        59. ])
```

In [ ]:

```
c= np.concatenate((a,b))
c
```

```
array([  0.5,   2. ,   4. ,   6. ,  10. ,  10.3,  12. ,  15. ,  16. ,
        18. ,  18. , 100. ,  10.5,   5. ,  15.6,   8. ,   0.5,  10.5,
       100.9,  15. ,  16. ,  59. ])
```

In [ ]:

```
c.sort()
```

```
c
```

```
array([  0.5,   0.5,   2. ,   4. ,   5. ,   6. ,   8. ,  10. ,  10.3,
        10.5,  10.5,  12. ,  15. ,  15. ,  15.6,  16. ,  16. ,  18. ,
        18. ,  59. , 100. , 100.9])
```

In [ ]:

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

```
array([[1, 2, 3],
       [2, 6, 5]])
```

In [ ]:

```
b = np.array ([[3,6,5],[6,8,9]])
b
```

```
array([[3, 6, 5],
       [6, 8, 9]])
```

In [ ]:

```
#checking the shape of matrix
b.shape
```

```
(2, 3)
```

In [ ]:

```
c= np.concatenate((a,b) ,axis= 1)
c
```

```
array([[1, 2, 3, 3, 6, 5],
       [2, 6, 5, 6, 8, 9]])
```

In [ ]:

```
c= np.concatenate((a,b) ,axis= 0)
c
```

```
array([[1, 2, 3],
       [2, 6, 5],
       [3, 6, 5],
       [6, 8, 9]])
```

In [ ]:

```
c.shape
```

```
(4, 3)
```

# 3-D Array

In [ ]:

```
a= np.array ([[['a','b','c'],['e','d','f']],
            [['a','b','c'],['e','d','f']],
            [['a','b','c'],['e','d','f']]])
a
```

```
array([[['a', 'b', 'c'],
        ['e', 'd', 'f']],

       [['a', 'b', 'c'],
        ['e', 'd', 'f']],

       [['a', 'b', 'c'],
        ['e', 'd', 'f']]], dtype='<U1')
```

In [ ]:

```
#finding a no. of dimensions
a.ndim
```

```
3
```

```
In [ ]: a.size
```

```
18
```

```
In [ ]: # shape of array
        a.shape
```

```
(3, 2, 3)
```

```
In [ ]: b= np.array ([[[1,2,3],[7,8,9],[9,6,3]],
                      [[1,2,3],[7,8,9],[9,6,3]],
                      [[1,2,3],[7,8,9],[9,6,3]]])
        b
```

```
array([[[1, 2, 3],
        [7, 8, 9],
        [9, 6, 3]],

       [[1, 2, 3],
        [7, 8, 9],
        [9, 6, 3]],

       [[1, 2, 3],
        [7, 8, 9],
        [9, 6, 3]]])
```

```
In [ ]: b.ndim
```

```
3
```

```
In [ ]: type(a)
```

```
numpy.ndarray
```

```
In [ ]: b.shape
```

```
(3, 3, 3)
```

```
In [ ]: b.size
```

```
27
```

converting 1d to 2d

```
In [ ]: a= np.arange(9)
        a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
In [ ]: a.reshape (3,3) #3*3=9 (9 indexes are there in array )
```

```
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
```

```
In [ ]: a.shape
```

```
(9,)
```

```
In [ ]: #row wise conversion
        b= a[np.newaxis,:]
```

```
b
```

```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8]])
```

In [ ]:
```
b.shape
```

```
(1, 9)
```

In [ ]:
```
#coloumn wise conversion
b= a[:, np.newaxis]
b
```

```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8]])
```

In [ ]:
```
b.shape
```

```
(9, 1)
```

In [ ]:
```
c= np.arange(9)
c
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [ ]:
```
c.shape
```

```
(9,)
```

In [ ]:
```
d=c[np.newaxis, :]
d
```

```
array([[0, 1, 2, 3, 4, 5, 6, 7, 8]])
```

In [ ]:
```
d.shape
```

```
(1, 9)
```

In [ ]:
```
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

In [ ]:
```
a[2]
```

```
2
```

In [ ]:
```
a[0:5]
```

```
array([0, 1, 2, 3, 4])
```

In [ ]:
```
a*6
```

```
array([ 0,  6, 12, 18, 24, 30, 36, 42, 48])
```

In [ ]:
```
a+6
```

```
array([ 6,  7,  8,  9, 10, 11, 12, 13, 14])
```

In [ ]:

```
a.sum()
```

```
36
```

In [ ]:

```
a.mean()
```

```
4.0
```

In [ ]:

```
a.max()
```

```
8
```

In [ ]:

```
a.min()
```

```
0
```

Pandas

How to install library

pip install pandas

pip instal numpy

Importing Libraries

In [ ]:

```
#importing libraries
import pandas as pd
import numpy as np
```

In [ ]:

```
# object creation
s= pd.Series([1,2,np.nan ,5,7,8,9])
s
```

```
0    1.0
1    2.0
2    NaN
3    5.0
4    7.0
5    8.0
6    9.0
dtype: float64
```

In [ ]:

```
dates = pd.date_range("20220101", periods=9)
dates
```

```
DatetimeIndex(['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
               '2022-01-05', '2022-01-06', '2022-01-07', '2022-01-08',
               '2022-01-09'],
              dtype='datetime64[ns]', freq='D')
```

In [ ]:

```
dates = pd.date_range("20220101", periods=33)
dates
```

```
DatetimeIndex(['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
               '2022-01-05', '2022-01-06', '2022-01-07', '2022-01-08',
               '2022-01-09', '2022-01-10', '2022-01-11', '2022-01-12',
               '2022-01-13', '2022-01-14', '2022-01-15', '2022-01-16',
               '2022-01-17', '2022-01-18', '2022-01-19', '2022-01-20',
               '2022-01-21', '2022-01-22', '2022-01-23', '2022-01-24',
               '2022-01-25', '2022-01-26', '2022-01-27', '2022-01-28',
               '2022-01-29', '2022-01-30', '2022-01-31', '2022-02-01',
               '2022-02-02'],
              dtype='datetime64[ns]', freq='D')
```

In [ ]:

```
df= pd.DataFrame(np.random.randn(33,5), index= dates, columns= list("ABCDE"))
df
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-06	-0.108873	-0.131410	1.177647	1.871027	0.957104
2022-01-07	-0.803777	-0.306501	0.564711	-0.089965	1.489706
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-09	-0.887664	0.049255	-1.409957	0.796757	0.045599
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454
2022-01-11	-0.373915	1.217292	1.359977	0.893264	1.218629
2022-01-12	-0.956893	-2.336328	0.175639	0.956212	-0.244871
2022-01-13	2.291410	-1.443694	0.618192	0.353503	1.150291
2022-01-14	-0.685916	0.657373	0.751282	1.633806	0.065106
2022-01-15	-1.863386	-1.267816	-0.591230	0.990356	-0.074070
2022-01-16	1.303350	0.472853	-0.789022	-1.022889	-0.159978
2022-01-17	0.320070	1.682127	0.482811	-0.487623	0.238106
2022-01-18	-0.509493	-1.134370	0.692725	-0.549689	-0.666717
2022-01-19	-1.726917	-0.766488	-0.221981	1.370493	-1.289682
2022-01-20	1.216959	1.642986	0.843236	-0.632844	-2.430845
2022-01-21	-0.849318	0.361485	-0.155557	0.835578	0.530294
2022-01-22	0.763374	-0.474104	0.068317	1.349778	-1.667274
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-24	-0.302701	-0.272333	1.918262	1.372899	0.178385
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-26	0.363980	-0.172259	-1.343963	1.476774	0.580969
2022-01-27	0.381062	0.845650	1.200135	-1.237464	1.457328
2022-01-28	0.986668	-0.771077	1.613589	-0.496968	0.033508
2022-01-29	-2.648280	0.757782	-0.349041	0.911459	-0.190662
2022-01-30	-0.003466	0.177465	0.218223	-0.373910	-0.888396
2022-01-31	0.045452	-0.666870	1.979702	0.244507	-0.088859
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196
2022-02-02	-2.107842	2.231678	-0.346340	0.337141	0.615166

In [ ]:

```
df2= pd.DataFrame(
    {
    "A" : 1.0,
    "B": pd.Timestamp("20130102"),
    "C": pd.Series(1, index= list(range(4)), dtype="float32"),
    "D": np.array([3]*4, dtype= "int32"),
    "E": pd.Categorical(["test","train","test","train"]),
    "F": "foo"
    }
    )
df2
```

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo
2	1.0	2013-01-02	1.0	3	test	foo
3	1.0	2013-01-02	1.0	3	train	foo

In [ ]:

```
df2.dtypes
```

```
A           float64
B    datetime64[ns]
C           float32
D             int32
E          category
F            object
dtype: object
```

In [ ]:

```
df2.head(2)
```

	A	B	C	D	E	F
0	1.0	2013-01-02	1.0	3	test	foo
1	1.0	2013-01-02	1.0	3	train	foo

In [ ]:

```
df.tail(2)
```

	A	B	C	D	E
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196
2022-02-02	-2.107842	2.231678	-0.346340	0.337141	0.615166

In [ ]:

```
df2.index
```

```
Int64Index([0, 1, 2, 3], dtype='int64')
```

In [ ]:

```
dates1 = pd.date_range("20220101", periods=20)
dates1
```

```
DatetimeIndex(['2022-01-01', '2022-01-02', '2022-01-03', '2022-01-04',
               '2022-01-05', '2022-01-06', '2022-01-07', '2022-01-08',
               '2022-01-09', '2022-01-10', '2022-01-11', '2022-01-12',
               '2022-01-13', '2022-01-14', '2022-01-15', '2022-01-16',
               '2022-01-17', '2022-01-18', '2022-01-19', '2022-01-20'],
              dtype='datetime64[ns]', freq='D')
```

In [1]:

```
df1= pd.DataFrame(np.random.randn(20,5), index= dates1, columns= list("ABCDE"))
df1
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
C:\Users\ADMINI~1\AppData\Local\Temp/ipykernel_7600/1674682586.py in <module>
----> 1 df1= pd.DataFrame(np.random.randn(20,5), index= dates1, columns= list("ABCDE"
))
      2 df1

NameError: name 'pd' is not defined
```

In [ ]:

```
a=df1.to_numpy()
```

In [ ]:

```
a.shape
```

```
(20, 5)
```

In [ ]:

```
df2.to_numpy()
```

```
array([[1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), 1.0, 3, 'train', 'foo']],
      dtype=object)
```

In [ ]:

```
#details of data
df1.describe()
```

	A	B	C	D	E
count	20.000000	20.000000	20.000000	20.000000	20.000000
mean	0.070508	-0.072575	0.115460	0.305385	-0.026136
std	0.975394	0.752431	1.036278	0.700803	0.697145
min	-2.090996	-1.646310	-1.909548	-1.152309	-1.371377
25%	-0.473375	-0.554341	-0.504664	-0.169159	-0.478535
50%	0.383890	-0.145480	-0.165726	0.218326	0.059924
75%	0.897546	0.641997	0.695434	0.690140	0.640159
max	1.694549	1.053939	1.992881	1.721367	0.917602

In [ ]:

```
#to transpose the data
df2.T
```

	0	1	2	3
A	1.0	1.0	1.0	1.0
B	2013-01-02 00:00:00	2013-01-02 00:00:00	2013-01-02 00:00:00	2013-01-02 00:00:00
C	1.0	1.0	1.0	1.0
D	3	3	3	3
E	test	train	test	train
F	foo	foo	foo	foo

In [ ]:
```
# Sorting
df1.sort_index(axis=0, ascending=False)
```

	A	B	C	D	E
2022-01-20	0.944428	-0.729589	1.992881	0.912663	-0.679367
2022-01-19	-0.906009	-0.012042	-0.190087	-0.161352	0.880138
2022-01-18	-0.498856	-1.646310	-1.909548	0.058233	-0.443734
2022-01-17	0.881919	0.283848	-0.200042	-0.216019	-0.762539
2022-01-16	0.322891	0.739421	0.476853	-0.594770	0.785129
2022-01-15	-0.616688	0.180063	1.008576	0.633678	-1.371377
2022-01-14	1.694549	-1.025499	1.584948	0.243552	0.309613
2022-01-13	0.467120	-0.405093	-1.485594	0.193101	0.164405
2022-01-12	-0.464881	0.992255	0.502942	0.436969	0.151128
2022-01-11	-2.090996	1.053939	-0.683397	1.721367	-0.331198
2022-01-10	0.955166	-0.193317	0.628846	-0.271304	0.917602
2022-01-09	-0.086092	-0.097643	0.342042	0.859524	0.874916
2022-01-08	-1.863741	0.727930	-0.141366	-0.192581	-0.582938
2022-01-07	-0.337541	-0.751140	1.933441	-0.081755	-0.362046
2022-01-06	0.963977	-0.716830	-0.381633	-1.152309	-1.178324
2022-01-05	0.469615	-0.458026	-0.454739	0.606912	0.040326
2022-01-04	-0.363039	0.613352	-0.495682	0.015500	0.713554
2022-01-03	0.444889	0.966369	-0.582827	0.440682	0.079521
2022-01-02	1.039896	-0.500178	-0.531612	1.393691	0.615694
2022-01-01	0.453561	-0.473015	0.895200	1.261925	-0.343220

In [ ]:
```
df.sort_values(by="B")
```

	A	B	C	D	E
2022-01-12	-0.956893	-2.336328	0.175639	0.956212	-0.244871
2022-01-13	2.291410	-1.443694	0.618192	0.353503	1.150291
2022-01-15	-1.863386	-1.267816	-0.591230	0.990356	-0.074070
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647
2022-01-18	-0.509493	-1.134370	0.692725	-0.549689	-0.666717
2022-01-28	0.986668	-0.771077	1.613589	-0.496968	0.033508
2022-01-19	-1.726917	-0.766488	-0.221981	1.370493	-1.289682
2022-01-31	0.045452	-0.666870	1.979702	0.244507	-0.088859
2022-01-22	0.763374	-0.474104	0.068317	1.349778	-1.667274
2022-01-07	-0.803777	-0.306501	0.564711	-0.089965	1.489706
2022-01-24	-0.302701	-0.272333	1.918262	1.372899	0.178385
2022-01-26	0.363980	-0.172259	-1.343963	1.476774	0.580969
2022-01-06	-0.108873	-0.131410	1.177647	1.871027	0.957104
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-09	-0.887664	0.049255	-1.409957	0.796757	0.045599
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196
2022-01-30	-0.003466	0.177465	0.218223	-0.373910	-0.888396
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-21	-0.849318	0.361485	-0.155557	0.835578	0.530294
2022-01-16	1.303350	0.472853	-0.789022	-1.022889	-0.159978
2022-01-14	-0.685916	0.657373	0.751282	1.633806	0.065106
2022-01-29	-2.648280	0.757782	-0.349041	0.911459	-0.190662
2022-01-27	0.381062	0.845650	1.200135	-1.237464	1.457328
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454
2022-01-11	-0.373915	1.217292	1.359977	0.893264	1.218629
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-20	1.216959	1.642986	0.843236	-0.632844	-2.430845
2022-01-17	0.320070	1.682127	0.482811	-0.487623	0.238106
2022-02-02	-2.107842	2.231678	-0.346340	0.337141	0.615166

```
In [ ]: df.sort_values(by="B",ascending=False)
```

	A	B	C	D	E
2022-02-02	-2.107842	2.231678	-0.346340	0.337141	0.615166
2022-01-17	0.320070	1.682127	0.482811	-0.487623	0.238106
2022-01-20	1.216959	1.642986	0.843236	-0.632844	-2.430845
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-11	-0.373915	1.217292	1.359977	0.893264	1.218629
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-27	0.381062	0.845650	1.200135	-1.237464	1.457328
2022-01-29	-2.648280	0.757782	-0.349041	0.911459	-0.190662
2022-01-14	-0.685916	0.657373	0.751282	1.633806	0.065106
2022-01-16	1.303350	0.472853	-0.789022	-1.022889	-0.159978
2022-01-21	-0.849318	0.361485	-0.155557	0.835578	0.530294
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-30	-0.003466	0.177465	0.218223	-0.373910	-0.888396
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196
2022-01-09	-0.887664	0.049255	-1.409957	0.796757	0.045599
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-06	-0.108873	-0.131410	1.177647	1.871027	0.957104
2022-01-26	0.363980	-0.172259	-1.343963	1.476774	0.580969
2022-01-24	-0.302701	-0.272333	1.918262	1.372899	0.178385
2022-01-07	-0.803777	-0.306501	0.564711	-0.089965	1.489706
2022-01-22	0.763374	-0.474104	0.068317	1.349778	-1.667274
2022-01-31	0.045452	-0.666870	1.979702	0.244507	-0.088859
2022-01-19	-1.726917	-0.766488	-0.221981	1.370493	-1.289682
2022-01-28	0.986668	-0.771077	1.613589	-0.496968	0.033508
2022-01-18	-0.509493	-1.134370	0.692725	-0.549689	-0.666717
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647
2022-01-15	-1.863386	-1.267816	-0.591230	0.990356	-0.074070
2022-01-13	2.291410	-1.443694	0.618192	0.353503	1.150291
2022-01-12	-0.956893	-2.336328	0.175639	0.956212	-0.244871

In [ ]:

```
df1["A"]
```

```
2022-01-01    0.453561
2022-01-02    1.039896
2022-01-03    0.444889
2022-01-04   -0.363039
2022-01-05    0.469615
2022-01-06    0.963977
2022-01-07   -0.337541
2022-01-08   -1.863741
2022-01-09   -0.086092
2022-01-10    0.955166
2022-01-11   -2.090996
2022-01-12   -0.464881
2022-01-13    0.467120
2022-01-14    1.694549
2022-01-15   -0.616688
2022-01-16    0.322891
2022-01-17    0.881919
2022-01-18   -0.498856
2022-01-19   -0.906009
2022-01-20    0.944428
Freq: D, Name: A, dtype: float64
```

In [ ]:

```
#filteration data with coloum wise or indexwise
df1["B"]
```

```
2022-01-01   -0.473015
2022-01-02   -0.500178
2022-01-03    0.966369
2022-01-04    0.613352
2022-01-05   -0.458026
2022-01-06   -0.716830
2022-01-07   -0.751140
2022-01-08    0.727930
2022-01-09   -0.097643
2022-01-10   -0.193317
2022-01-11    1.053939
2022-01-12    0.992255
2022-01-13   -0.405093
2022-01-14   -1.025499
2022-01-15    0.180063
2022-01-16    0.739421
2022-01-17    0.283848
2022-01-18   -1.646310
2022-01-19   -0.012042
2022-01-20   -0.729589
Freq: D, Name: B, dtype: float64
```

In [ ]:

```
# TO select data row wise
df[0:1]
```

	A	B	C	D	E
2022-01-01	0.10504	1.243529	0.047028	0.342738	-0.837202

In [ ]:

```
df[0:2]
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214

```
In [ ]: df[0:10]
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-06	-0.108873	-0.131410	1.177647	1.871027	0.957104
2022-01-07	-0.803777	-0.306501	0.564711	-0.089965	1.489706
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-09	-0.887664	0.049255	-1.409957	0.796757	0.045599
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454

```
In [ ]: df[1:10]
```

	A	B	C	D	E
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-06	-0.108873	-0.131410	1.177647	1.871027	0.957104
2022-01-07	-0.803777	-0.306501	0.564711	-0.089965	1.489706
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-09	-0.887664	0.049255	-1.409957	0.796757	0.045599
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454

```
In [ ]: df1.head()
```

	A	B	C	D	E
2022-01-01	0.453561	-0.473015	0.895200	1.261925	-0.343220
2022-01-02	1.039896	-0.500178	-0.531612	1.393691	0.615694
2022-01-03	0.444889	0.966369	-0.582827	0.440682	0.079521
2022-01-04	-0.363039	0.613352	-0.495682	0.015500	0.713554
2022-01-05	0.469615	-0.458026	-0.454739	0.606912	0.040326

In [ ]:

```
# showing the only 16th row  and its values
df.loc[dates[15]]
```

```
A    1.303350
B    0.472853
C   -0.789022
D   -1.022889
E   -0.159978
Name: 2022-01-16 00:00:00, dtype: float64
```

In [ ]:

```
#multiple axis lables
df.loc[:, ["A","B"]]
```

	A	B
2022-01-01	0.105040	1.243529
2022-01-02	1.154318	-0.089813
2022-01-03	0.421724	0.236177
2022-01-04	-1.275176	-1.191302
2022-01-05	0.821600	-0.090374
2022-01-06	-0.108873	-0.131410
2022-01-07	-0.803777	-0.306501
2022-01-08	1.066226	1.446267
2022-01-09	-0.887664	0.049255
2022-01-10	2.008289	0.989671
2022-01-11	-0.373915	1.217292
2022-01-12	-0.956893	-2.336328
2022-01-13	2.291410	-1.443694
2022-01-14	-0.685916	0.657373
2022-01-15	-1.863386	-1.267816
2022-01-16	1.303350	0.472853
2022-01-17	0.320070	1.682127
2022-01-18	-0.509493	-1.134370
2022-01-19	-1.726917	-0.766488
2022-01-20	1.216959	1.642986
2022-01-21	-0.849318	0.361485
2022-01-22	0.763374	-0.474104
2022-01-23	0.374633	0.926311
2022-01-24	-0.302701	-0.272333
2022-01-25	1.853834	0.326428
2022-01-26	0.363980	-0.172259
2022-01-27	0.381062	0.845650
2022-01-28	0.986668	-0.771077
2022-01-29	-2.648280	0.757782
2022-01-30	-0.003466	0.177465
2022-01-31	0.045452	-0.666870
2022-02-01	1.149844	0.146726
2022-02-02	-2.107842	2.231678

In [ ]:

```
df.loc["20220109":"20220113",["A","B", "C"]]
```

	A	B	C
2022-01-09	-0.887664	0.049255	-1.409957
2022-01-10	2.008289	0.989671	-0.996114
2022-01-11	-0.373915	1.217292	1.359977
2022-01-12	-0.956893	-2.336328	0.175639
2022-01-13	2.291410	-1.443694	0.618192

In [ ]:

```
df.loc["20220109",["A","B", "C"]]
```

```
A   -0.887664
B    0.049255
C   -1.409957
Name: 2022-01-09 00:00:00, dtype: float64
```

In [ ]:

```
#Scalar value
df.at[dates[0],"A"]
```

```
0.10504008812710756
```

In [ ]:

```
df.iloc[3]
```

```
A   -1.275176
B   -1.191302
C   -1.247612
D    0.670763
E    0.568647
Name: 2022-01-04 00:00:00, dtype: float64
```

In [ ]:

```
df.iloc[3:10]
```

	A	B	C	D	E
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-06	-0.108873	-0.131410	1.177647	1.871027	0.957104
2022-01-07	-0.803777	-0.306501	0.564711	-0.089965	1.489706
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-09	-0.887664	0.049255	-1.409957	0.796757	0.045599
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454

In [ ]:

```
#       rows col
df.iloc[0:5, 0:2]
```

	A	B
2022-01-01	0.105040	1.243529
2022-01-02	1.154318	-0.089813
2022-01-03	0.421724	0.236177
2022-01-04	-1.275176	-1.191302
2022-01-05	0.821600	-0.090374

In [ ]:

```
df.iloc[:, 0:2]
```

	A	B
2022-01-01	0.105040	1.243529
2022-01-02	1.154318	-0.089813
2022-01-03	0.421724	0.236177
2022-01-04	-1.275176	-1.191302
2022-01-05	0.821600	-0.090374
2022-01-06	-0.108873	-0.131410
2022-01-07	-0.803777	-0.306501
2022-01-08	1.066226	1.446267
2022-01-09	-0.887664	0.049255
2022-01-10	2.008289	0.989671
2022-01-11	-0.373915	1.217292
2022-01-12	-0.956893	-2.336328
2022-01-13	2.291410	-1.443694
2022-01-14	-0.685916	0.657373
2022-01-15	-1.863386	-1.267816
2022-01-16	1.303350	0.472853
2022-01-17	0.320070	1.682127
2022-01-18	-0.509493	-1.134370
2022-01-19	-1.726917	-0.766488
2022-01-20	1.216959	1.642986
2022-01-21	-0.849318	0.361485
2022-01-22	0.763374	-0.474104
2022-01-23	0.374633	0.926311
2022-01-24	-0.302701	-0.272333
2022-01-25	1.853834	0.326428
2022-01-26	0.363980	-0.172259
2022-01-27	0.381062	0.845650
2022-01-28	0.986668	-0.771077
2022-01-29	-2.648280	0.757782
2022-01-30	-0.003466	0.177465
2022-01-31	0.045452	-0.666870
2022-02-01	1.149844	0.146726
2022-02-02	-2.107842	2.231678

In [ ]:

```
df[df["A"]> 0]
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454
2022-01-13	2.291410	-1.443694	0.618192	0.353503	1.150291
2022-01-16	1.303350	0.472853	-0.789022	-1.022889	-0.159978
2022-01-17	0.320070	1.682127	0.482811	-0.487623	0.238106
2022-01-20	1.216959	1.642986	0.843236	-0.632844	-2.430845
2022-01-22	0.763374	-0.474104	0.068317	1.349778	-1.667274
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-26	0.363980	-0.172259	-1.343963	1.476774	0.580969
2022-01-27	0.381062	0.845650	1.200135	-1.237464	1.457328
2022-01-28	0.986668	-0.771077	1.613589	-0.496968	0.033508
2022-01-31	0.045452	-0.666870	1.979702	0.244507	-0.088859
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196

In [ ]:

```
df[df["A"]> 0]
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454
2022-01-13	2.291410	-1.443694	0.618192	0.353503	1.150291
2022-01-16	1.303350	0.472853	-0.789022	-1.022889	-0.159978
2022-01-17	0.320070	1.682127	0.482811	-0.487623	0.238106
2022-01-20	1.216959	1.642986	0.843236	-0.632844	-2.430845
2022-01-22	0.763374	-0.474104	0.068317	1.349778	-1.667274
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-26	0.363980	-0.172259	-1.343963	1.476774	0.580969
2022-01-27	0.381062	0.845650	1.200135	-1.237464	1.457328
2022-01-28	0.986668	-0.771077	1.613589	-0.496968	0.033508
2022-01-31	0.045452	-0.666870	1.979702	0.244507	-0.088859
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196

Assignment

```
In [ ]:
# assignment :Getting non zero values in more then one column
df[ df.iloc[:, 0:5]> 0]
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	NaN
2022-01-02	1.154318	NaN	NaN	NaN	NaN
2022-01-03	0.421724	0.236177	NaN	0.103659	0.831399
2022-01-04	NaN	NaN	NaN	0.670763	0.568647
2022-01-05	0.821600	NaN	NaN	0.939088	NaN
2022-01-06	NaN	NaN	1.177647	1.871027	0.957104
2022-01-07	NaN	NaN	0.564711	NaN	1.489706
2022-01-08	1.066226	1.446267	NaN	NaN	NaN
2022-01-09	NaN	0.049255	NaN	0.796757	0.045599
2022-01-10	2.008289	0.989671	NaN	NaN	0.734454
2022-01-11	NaN	1.217292	1.359977	0.893264	1.218629
2022-01-12	NaN	NaN	0.175639	0.956212	NaN
2022-01-13	2.291410	NaN	0.618192	0.353503	1.150291
2022-01-14	NaN	0.657373	0.751282	1.633806	0.065106
2022-01-15	NaN	NaN	NaN	0.990356	NaN
2022-01-16	1.303350	0.472853	NaN	NaN	NaN
2022-01-17	0.320070	1.682127	0.482811	NaN	0.238106
2022-01-18	NaN	NaN	0.692725	NaN	NaN
2022-01-19	NaN	NaN	NaN	1.370493	NaN
2022-01-20	1.216959	1.642986	0.843236	NaN	NaN
2022-01-21	NaN	0.361485	NaN	0.835578	0.530294
2022-01-22	0.763374	NaN	0.068317	1.349778	NaN
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-24	NaN	NaN	1.918262	1.372899	0.178385
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-26	0.363980	NaN	NaN	1.476774	0.580969
2022-01-27	0.381062	0.845650	1.200135	NaN	1.457328
2022-01-28	0.986668	NaN	1.613589	NaN	0.033508
2022-01-29	NaN	0.757782	NaN	0.911459	NaN
2022-01-30	NaN	0.177465	0.218223	NaN	NaN
2022-01-31	0.045452	NaN	1.979702	0.244507	NaN
2022-02-01	1.149844	0.146726	0.001847	NaN	0.252196
2022-02-02	NaN	2.231678	NaN	0.337141	0.615166

In [ ]:

```
df[df["A"]> 0]
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454
2022-01-13	2.291410	-1.443694	0.618192	0.353503	1.150291
2022-01-16	1.303350	0.472853	-0.789022	-1.022889	-0.159978
2022-01-17	0.320070	1.682127	0.482811	-0.487623	0.238106
2022-01-20	1.216959	1.642986	0.843236	-0.632844	-2.430845
2022-01-22	0.763374	-0.474104	0.068317	1.349778	-1.667274
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-26	0.363980	-0.172259	-1.343963	1.476774	0.580969
2022-01-27	0.381062	0.845650	1.200135	-1.237464	1.457328
2022-01-28	0.986668	-0.771077	1.613589	-0.496968	0.033508
2022-01-31	0.045452	-0.666870	1.979702	0.244507	-0.088859
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196

In [ ]:

```
df[df>0]
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	NaN
2022-01-02	1.154318	NaN	NaN	NaN	NaN
2022-01-03	0.421724	0.236177	NaN	0.103659	0.831399
2022-01-04	NaN	NaN	NaN	0.670763	0.568647
2022-01-05	0.821600	NaN	NaN	0.939088	NaN
2022-01-06	NaN	NaN	1.177647	1.871027	0.957104
2022-01-07	NaN	NaN	0.564711	NaN	1.489706
2022-01-08	1.066226	1.446267	NaN	NaN	NaN
2022-01-09	NaN	0.049255	NaN	0.796757	0.045599
2022-01-10	2.008289	0.989671	NaN	NaN	0.734454
2022-01-11	NaN	1.217292	1.359977	0.893264	1.218629
2022-01-12	NaN	NaN	0.175639	0.956212	NaN
2022-01-13	2.291410	NaN	0.618192	0.353503	1.150291
2022-01-14	NaN	0.657373	0.751282	1.633806	0.065106
2022-01-15	NaN	NaN	NaN	0.990356	NaN
2022-01-16	1.303350	0.472853	NaN	NaN	NaN
2022-01-17	0.320070	1.682127	0.482811	NaN	0.238106
2022-01-18	NaN	NaN	0.692725	NaN	NaN
2022-01-19	NaN	NaN	NaN	1.370493	NaN
2022-01-20	1.216959	1.642986	0.843236	NaN	NaN
2022-01-21	NaN	0.361485	NaN	0.835578	0.530294
2022-01-22	0.763374	NaN	0.068317	1.349778	NaN
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-24	NaN	NaN	1.918262	1.372899	0.178385
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-26	0.363980	NaN	NaN	1.476774	0.580969
2022-01-27	0.381062	0.845650	1.200135	NaN	1.457328
2022-01-28	0.986668	NaN	1.613589	NaN	0.033508
2022-01-29	NaN	0.757782	NaN	0.911459	NaN
2022-01-30	NaN	0.177465	0.218223	NaN	NaN
2022-01-31	0.045452	NaN	1.979702	0.244507	NaN
2022-02-01	1.149844	0.146726	0.001847	NaN	0.252196
2022-02-02	NaN	2.231678	NaN	0.337141	0.615166

In [ ]:

```
df[ df.iloc[0:3]> 0 ]
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	NaN
2022-01-02	1.154318	NaN	NaN	NaN	NaN
2022-01-03	0.421724	0.236177	NaN	0.103659	0.831399
2022-01-04	NaN	NaN	NaN	NaN	NaN
2022-01-05	NaN	NaN	NaN	NaN	NaN
2022-01-06	NaN	NaN	NaN	NaN	NaN
2022-01-07	NaN	NaN	NaN	NaN	NaN
2022-01-08	NaN	NaN	NaN	NaN	NaN
2022-01-09	NaN	NaN	NaN	NaN	NaN
2022-01-10	NaN	NaN	NaN	NaN	NaN
2022-01-11	NaN	NaN	NaN	NaN	NaN
2022-01-12	NaN	NaN	NaN	NaN	NaN
2022-01-13	NaN	NaN	NaN	NaN	NaN
2022-01-14	NaN	NaN	NaN	NaN	NaN
2022-01-15	NaN	NaN	NaN	NaN	NaN
2022-01-16	NaN	NaN	NaN	NaN	NaN
2022-01-17	NaN	NaN	NaN	NaN	NaN
2022-01-18	NaN	NaN	NaN	NaN	NaN
2022-01-19	NaN	NaN	NaN	NaN	NaN
2022-01-20	NaN	NaN	NaN	NaN	NaN
2022-01-21	NaN	NaN	NaN	NaN	NaN
2022-01-22	NaN	NaN	NaN	NaN	NaN
2022-01-23	NaN	NaN	NaN	NaN	NaN
2022-01-24	NaN	NaN	NaN	NaN	NaN
2022-01-25	NaN	NaN	NaN	NaN	NaN
2022-01-26	NaN	NaN	NaN	NaN	NaN
2022-01-27	NaN	NaN	NaN	NaN	NaN
2022-01-28	NaN	NaN	NaN	NaN	NaN
2022-01-29	NaN	NaN	NaN	NaN	NaN
2022-01-30	NaN	NaN	NaN	NaN	NaN
2022-01-31	NaN	NaN	NaN	NaN	NaN
2022-02-01	NaN	NaN	NaN	NaN	NaN
2022-02-02	NaN	NaN	NaN	NaN	NaN

In [ ]:

```
# IS IN Method
df3 = df.copy()
df3
```

	A	B	C	D	E
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521
2022-01-06	-0.108873	-0.131410	1.177647	1.871027	0.957104
2022-01-07	-0.803777	-0.306501	0.564711	-0.089965	1.489706
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464
2022-01-09	-0.887664	0.049255	-1.409957	0.796757	0.045599
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454
2022-01-11	-0.373915	1.217292	1.359977	0.893264	1.218629
2022-01-12	-0.956893	-2.336328	0.175639	0.956212	-0.244871
2022-01-13	2.291410	-1.443694	0.618192	0.353503	1.150291
2022-01-14	-0.685916	0.657373	0.751282	1.633806	0.065106
2022-01-15	-1.863386	-1.267816	-0.591230	0.990356	-0.074070
2022-01-16	1.303350	0.472853	-0.789022	-1.022889	-0.159978
2022-01-17	0.320070	1.682127	0.482811	-0.487623	0.238106
2022-01-18	-0.509493	-1.134370	0.692725	-0.549689	-0.666717
2022-01-19	-1.726917	-0.766488	-0.221981	1.370493	-1.289682
2022-01-20	1.216959	1.642986	0.843236	-0.632844	-2.430845
2022-01-21	-0.849318	0.361485	-0.155557	0.835578	0.530294
2022-01-22	0.763374	-0.474104	0.068317	1.349778	-1.667274
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480
2022-01-24	-0.302701	-0.272333	1.918262	1.372899	0.178385
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401
2022-01-26	0.363980	-0.172259	-1.343963	1.476774	0.580969
2022-01-27	0.381062	0.845650	1.200135	-1.237464	1.457328
2022-01-28	0.986668	-0.771077	1.613589	-0.496968	0.033508
2022-01-29	-2.648280	0.757782	-0.349041	0.911459	-0.190662
2022-01-30	-0.003466	0.177465	0.218223	-0.373910	-0.888396
2022-01-31	0.045452	-0.666870	1.979702	0.244507	-0.088859
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196
2022-02-02	-2.107842	2.231678	-0.346340	0.337141	0.615166

In [ ]:

```
#Adding a column
df3["BABA"]= [1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,1,2,3,4,5,6,7,8,9,10,1,2,3]
df3
```

	A	B	C	D	E	BABA
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202	1
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214	2
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399	3
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647	4
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521	5
2022-01-06	-0.108873	-0.131410	1.177647	1.871027	0.957104	6
2022-01-07	-0.803777	-0.306501	0.564711	-0.089965	1.489706	7
2022-01-08	1.066226	1.446267	-0.641866	-0.812720	-1.891464	8
2022-01-09	-0.887664	0.049255	-1.409957	0.796757	0.045599	9
2022-01-10	2.008289	0.989671	-0.996114	-1.210864	0.734454	10
2022-01-11	-0.373915	1.217292	1.359977	0.893264	1.218629	1
2022-01-12	-0.956893	-2.336328	0.175639	0.956212	-0.244871	2
2022-01-13	2.291410	-1.443694	0.618192	0.353503	1.150291	3
2022-01-14	-0.685916	0.657373	0.751282	1.633806	0.065106	4
2022-01-15	-1.863386	-1.267816	-0.591230	0.990356	-0.074070	5
2022-01-16	1.303350	0.472853	-0.789022	-1.022889	-0.159978	6
2022-01-17	0.320070	1.682127	0.482811	-0.487623	0.238106	7
2022-01-18	-0.509493	-1.134370	0.692725	-0.549689	-0.666717	8
2022-01-19	-1.726917	-0.766488	-0.221981	1.370493	-1.289682	9
2022-01-20	1.216959	1.642986	0.843236	-0.632844	-2.430845	10
2022-01-21	-0.849318	0.361485	-0.155557	0.835578	0.530294	1
2022-01-22	0.763374	-0.474104	0.068317	1.349778	-1.667274	2
2022-01-23	0.374633	0.926311	1.621113	0.756842	1.283480	3
2022-01-24	-0.302701	-0.272333	1.918262	1.372899	0.178385	4
2022-01-25	1.853834	0.326428	2.222420	0.870315	0.727401	5
2022-01-26	0.363980	-0.172259	-1.343963	1.476774	0.580969	6
2022-01-27	0.381062	0.845650	1.200135	-1.237464	1.457328	7
2022-01-28	0.986668	-0.771077	1.613589	-0.496968	0.033508	8
2022-01-29	-2.648280	0.757782	-0.349041	0.911459	-0.190662	9
2022-01-30	-0.003466	0.177465	0.218223	-0.373910	-0.888396	10
2022-01-31	0.045452	-0.666870	1.979702	0.244507	-0.088859	1
2022-02-01	1.149844	0.146726	0.001847	-1.705877	0.252196	2
2022-02-02	-2.107842	2.231678	-0.346340	0.337141	0.615166	3

In [ ]:

```
#Adding a new column having same value of previous column
df3["Mean"] =df3["A"]
df3.head()
```

	A	B	C	D	E	BABA	Mean
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202	1	0.105040
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214	2	1.154318
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399	3	0.421724
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647	4	-1.275176
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521	5	0.821600

Assignment

In [ ]:

```
#Adding a column having mean of previous values
# #Assignment no 2
df3["Mean"] =df3.mean(axis= 1)
df3.head()
```

	A	B	C	D	E	BABA	Mean
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202	1	0.286596
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214	2	0.287786
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399	3	0.664494
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647	4	0.035735
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521	5	0.799945

In [ ]:

```
df["New"]="new hai "
df.head()
```

	A	B	C	D	E	New
2022-01-01	0.105040	1.243529	0.047028	0.342738	-0.837202	new hai
2022-01-02	1.154318	-0.089813	-1.416921	-0.595189	-0.192214	new hai
2022-01-03	0.421724	0.236177	-0.363225	0.103659	0.831399	new hai
2022-01-04	-1.275176	-1.191302	-1.247612	0.670763	0.568647	new hai
2022-01-05	0.821600	-0.090374	-1.038776	0.939088	-0.853521	new hai

In [ ]:

```
df.insert(2,"Beech me ", "18+")
df.head()
```

	A	B	Beech me	C	D	E	New
2022-01-01	0.105040	1.243529	18+	0.047028	0.342738	-0.837202	new hai
2022-01-02	1.154318	-0.089813	18+	-1.416921	-0.595189	-0.192214	new hai
2022-01-03	0.421724	0.236177	18+	-0.363225	0.103659	0.831399	new hai
2022-01-04	-1.275176	-1.191302	18+	-1.247612	0.670763	0.568647	new hai
2022-01-05	0.821600	-0.090374	18+	-1.038776	0.939088	-0.853521	new hai

In [ ]:

```
df["Concatinated"]= df["A"]+df["B"]
df.head()
```

	A	B	Beech me	C	D	E	New	Concatinated
2022-01-01	0.105040	1.243529	18+	0.047028	0.342738	-0.837202	new hai	1.348569
2022-01-02	1.154318	-0.089813	18+	-1.416921	-0.595189	-0.192214	new hai	1.064505
2022-01-03	0.421724	0.236177	18+	-0.363225	0.103659	0.831399	new hai	0.657901
2022-01-04	-1.275176	-1.191302	18+	-1.247612	0.670763	0.568647	new hai	-2.466478
2022-01-05	0.821600	-0.090374	18+	-1.038776	0.939088	-0.853521	new hai	0.731226

Working on FAOSTAT website data

Population of Countries (1950 to 2018)

In this project we create plots related to the increase in population over the years for different countries.

> Step -1 : Importing Libraries and calling the csv file

In [ ]:

```
# Importing libraries

import pandas as pd
import numpy as np

# Loading the data base of population

p_data = pd.read_csv("FAOSTAT_data_1-12-2022.csv")
p_data
```

	Domain Code	Domain	Area Code (FAO)	Area	Element Code	Element	Item Code	Item	Year Code	Year
0	OA	Annual population	2	Afghanistan	511	Total Population - Both sexes	3010	Population - Est. & Proj.	1950	1950
1	OA	Annual population	2	Afghanistan	512	Total Population - Male	3010	Population - Est. & Proj.	1950	1950
2	OA	Annual population	2	Afghanistan	513	Total Population - Female	3010	Population - Est. & Proj.	1950	1950
3	OA	Annual population	2	Afghanistan	551	Rural population	3010	Population - Est. & Proj.	1950	1950
4	OA	Annual population	2	Afghanistan	561	Urban population	3010	Population - Est. & Proj.	1950	1950
...	...	...	...	...	...	...	...	...	...	...
70580	OA	Annual population	181	Zimbabwe	511	Total Population - Both sexes	3010	Population - Est. & Proj.	2018	2018
70581	OA	Annual population	181	Zimbabwe	512	Total Population - Male	3010	Population - Est. & Proj.	2018	2018
70582	OA	Annual population	181	Zimbabwe	513	Total Population - Female	3010	Population - Est. & Proj.	2018	2018
70583	OA	Annual population	181	Zimbabwe	551	Rural population	3010	Population - Est. & Proj.	2018	2018
70584	OA	Annual population	181	Zimbabwe	561	Urban population	3010	Population - Est. & Proj.	2018	2018

70585 rows × 16 columns

> Step-2 : Filtering out the columns

In [ ]:
```
df = pd.DataFrame(p_data, columns=["Area Code (FAO)","Area","Year","value","Element"])
df
```

	Area Code (FAO)	Area	Year	value	Element
0	2	Afghanistan	1950	7752118	Total Population - Both sexes
1	2	Afghanistan	1950	4099243	Total Population - Male
2	2	Afghanistan	1950	3652874	Total Population - Female
3	2	Afghanistan	1950	7286991	Rural population
4	2	Afghanistan	1950	465127	Urban population
...	...	...	...	...	...
70580	181	Zimbabwe	2018	14438802	Total Population - Both sexes
70581	181	Zimbabwe	2018	6879119	Total Population - Male
70582	181	Zimbabwe	2018	7559693	Total Population - Female
70583	181	Zimbabwe	2018	11465748	Rural population
70584	181	Zimbabwe	2018	5447513	Urban population

70585 rows × 5 columns

> Step-3 : Changing Index to Elements so that we can further clean the data according to our requirements.

In [ ]:
```
df1 = df.set_index("Element")
df1
```

	Area Code (FAO)	Area	Year	value
Element				
Total Population - Both sexes	2	Afghanistan	1950	7752118
Total Population - Male	2	Afghanistan	1950	4099243
Total Population - Female	2	Afghanistan	1950	3652874
Rural population	2	Afghanistan	1950	7286991
Urban population	2	Afghanistan	1950	465127
...	...	...	...	...
Total Population - Both sexes	181	Zimbabwe	2018	14438802
Total Population - Male	181	Zimbabwe	2018	6879119
Total Population - Female	181	Zimbabwe	2018	7559693
Rural population	181	Zimbabwe	2018	11465748
Urban population	181	Zimbabwe	2018	5447513

70585 rows × 4 columns

> Step-4 : Eliminating those indexes which contains additional information

In [ ]:

```
df2 = df1.loc["Total Population - Both sexes"]
df2
```

	Area Code (FAO)	Area	Year	value
Element				
Total Population - Both sexes	2	Afghanistan	1950	7752118
Total Population - Both sexes	2	Afghanistan	1951	7840156
Total Population - Both sexes	2	Afghanistan	1952	7935997
Total Population - Both sexes	2	Afghanistan	1953	8039694
Total Population - Both sexes	2	Afghanistan	1954	8151317
...	...	...	...	...
Total Population - Both sexes	181	Zimbabwe	2014	13586707
Total Population - Both sexes	181	Zimbabwe	2015	13814629
Total Population - Both sexes	181	Zimbabwe	2016	14030331
Total Population - Both sexes	181	Zimbabwe	2017	14236595
Total Population - Both sexes	181	Zimbabwe	2018	14438802

14915 rows × 4 columns

> Step-5 : Clean the data set further so that we can filter out further the information we required.

In [ ]:

```
df3 = df2.set_index("Area")
df3
```

	Area Code (FAO)	Year	value
Area			
Afghanistan	2	1950	7752118
Afghanistan	2	1951	7840156
Afghanistan	2	1952	7935997
Afghanistan	2	1953	8039694
Afghanistan	2	1954	8151317
...	...	...	...
Zimbabwe	181	2014	13586707
Zimbabwe	181	2015	13814629
Zimbabwe	181	2016	14030331
Zimbabwe	181	2017	14236595
Zimbabwe	181	2018	14438802

14915 rows × 3 columns

> Step - 6 : Calling out the required country data

In [ ]:

```
df4 = df3.loc["Pakistan"]
df4
```

	Area Code (FAO)	Year	value
Area			
Pakistan	165	1950	37542376
Pakistan	165	1951	37992886
Pakistan	165	1952	38516515
Pakistan	165	1953	39109093
Pakistan	165	1954	39767174
...	...	...	...
Pakistan	165	2014	195305013
Pakistan	165	2015	199426964
Pakistan	165	2016	203631353
Pakistan	165	2017	207906209
Pakistan	165	2018	212228286

69 rows × 3 columns

> Step- 7 : Creating Plots

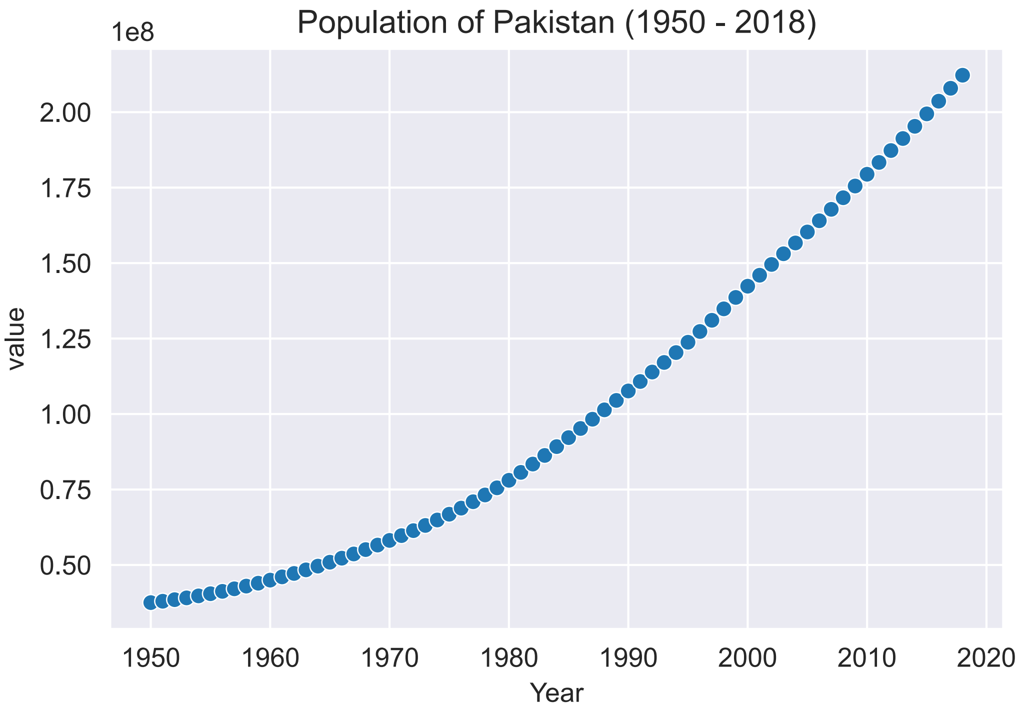
In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("darkgrid")

plot_pop = sns.scatterplot(data=df4, x="Year", y="value")
plt.title("Population of Pakistan (1950 - 2018)")
plot_pop
```

```
<AxesSubplot:title={'center':'Population of Pakistan (1950 - 2018)'}, xlabel='Year',
ylabel='value'>
```



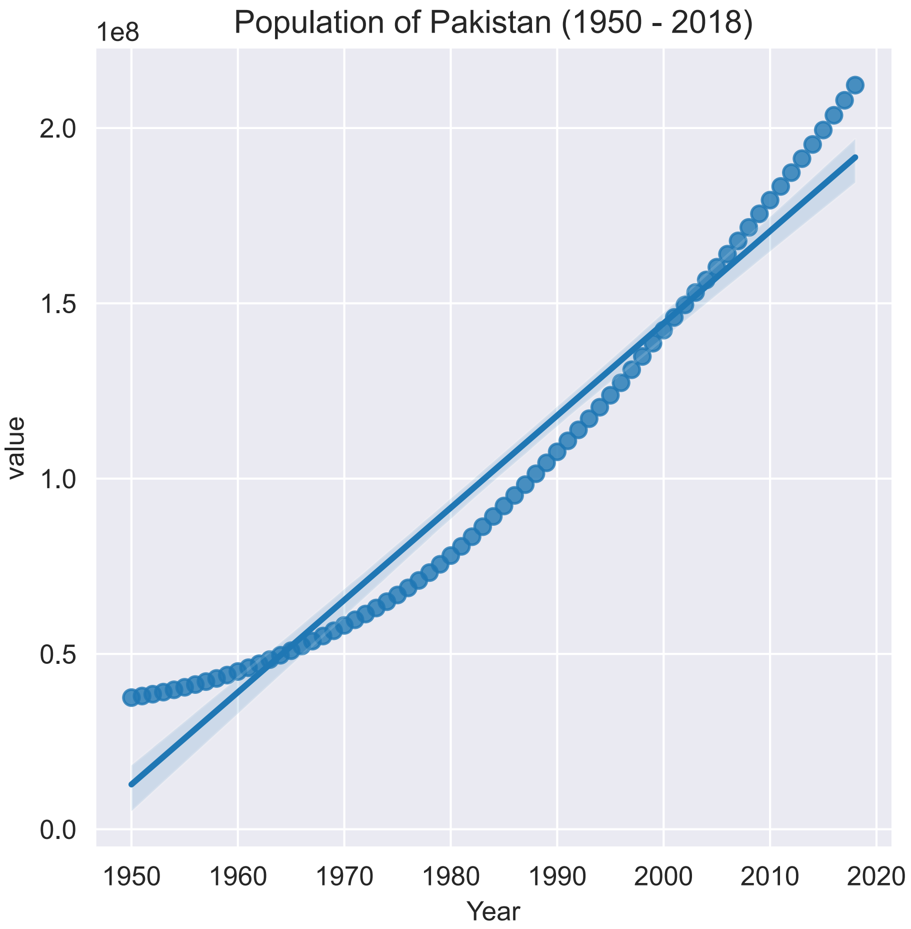
In [ ]:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("darkgrid")

plot_pop = sns.lmplot(data=df4, x="Year", y="value")
plt.title("Population of Pakistan (1950 - 2018)")
plot_pop
```

```
<seaborn.axisgrid.FacetGrid at 0x211213203a0>
```



Population Trend

This project shows the rate of increase in the population of male, female, rural area and urban areas of Pakistan

```
In [ ]: df5 = df.set_index("Area Code (FAO)")
        df5
```

	Area	Year	value	Element
Area Code (FAO)				
2	Afghanistan	1950	7752118	Total Population - Both sexes
2	Afghanistan	1950	4099243	Total Population - Male
2	Afghanistan	1950	3652874	Total Population - Female
2	Afghanistan	1950	7286991	Rural population
2	Afghanistan	1950	465127	Urban population
...	...	...	...	...
181	Zimbabwe	2018	14438802	Total Population - Both sexes
181	Zimbabwe	2018	6879119	Total Population - Male
181	Zimbabwe	2018	7559693	Total Population - Female
181	Zimbabwe	2018	11465748	Rural population
181	Zimbabwe	2018	5447513	Urban population

70585 rows × 4 columns

```
In [ ]: df6 = df5.loc[165]
        df6
```

	Area	Year	value	Element
Area Code (FAO)				
165	Pakistan	1950	37542376	Total Population - Both sexes
165	Pakistan	1950	20461235	Total Population - Male
165	Pakistan	1950	17081135	Total Population - Female
165	Pakistan	1950	30964622	Rural population
165	Pakistan	1950	6577754	Urban population
...	...	...	...	...
165	Pakistan	2018	212228286	Total Population - Both sexes
165	Pakistan	2018	109216763	Total Population - Male
165	Pakistan	2018	103011525	Total Population - Female
165	Pakistan	2018	127183388	Rural population
165	Pakistan	2018	73630430	Urban population

345 rows × 4 columns

```
In [ ]: df7 = df6.set_index("Element")
        df7
```

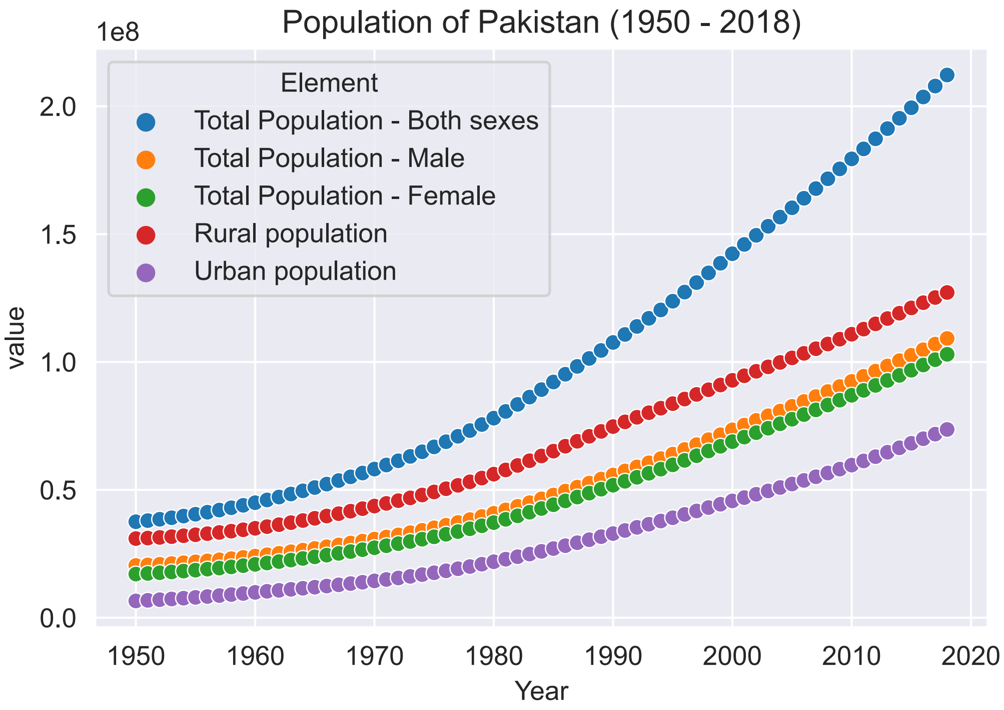
	Area	Year	value
Element			
Total Population - Both sexes	Pakistan	1950	37542376
Total Population - Male	Pakistan	1950	20461235
Total Population - Female	Pakistan	1950	17081135
Rural population	Pakistan	1950	30964622
Urban population	Pakistan	1950	6577754
...	...	...	...
Total Population - Both sexes	Pakistan	2018	212228286
Total Population - Male	Pakistan	2018	109216763
Total Population - Female	Pakistan	2018	103011525
Rural population	Pakistan	2018	127183388
Urban population	Pakistan	2018	73630430

345 rows × 3 columns

```
In [ ]: sns.set_style("darkgrid")

        plot2 = sns.scatterplot(data= df7, x="Year", y="value", hue="Element")
        plt.title("Population of Pakistan (1950 - 2018)")
        plot2
```

```
<AxesSubplot:title={'center':'Population of Pakistan (1950 - 2018)'}, xlabel='Year',
ylabel='value'>
```



Population of Countries in 2018

In this project we compare the population of different countries

In [ ]:
```python
df2
```

	Area Code (FAO)	Area	Year	value
Element				
Total Population - Both sexes	2	Afghanistan	1950	7752118
Total Population - Both sexes	2	Afghanistan	1951	7840156
Total Population - Both sexes	2	Afghanistan	1952	7935997
Total Population - Both sexes	2	Afghanistan	1953	8039694
Total Population - Both sexes	2	Afghanistan	1954	8151317
...	...	...	...	...
Total Population - Both sexes	181	Zimbabwe	2014	13586707
Total Population - Both sexes	181	Zimbabwe	2015	13814629
Total Population - Both sexes	181	Zimbabwe	2016	14030331
Total Population - Both sexes	181	Zimbabwe	2017	14236595
Total Population - Both sexes	181	Zimbabwe	2018	14438802

14915 rows × 4 columns

In [ ]:
```python
df8 = df2[df2["Year"] == 2018]
df8
```

	Area Code (FAO)	Area	Year	value
Element				
Total Population - Both sexes	2	Afghanistan	2018	37171921
Total Population - Both sexes	3	Albania	2018	2882740
Total Population - Both sexes	4	Algeria	2018	42228408
Total Population - Both sexes	5	American Samoa	2018	55465
Total Population - Both sexes	6	Andorra	2018	77006
...	...	...	...	...
Total Population - Both sexes	243	Wallis and Futuna Islands	2018	11661
Total Population - Both sexes	205	Western Sahara	2018	567402
Total Population - Both sexes	249	Yemen	2018	28498683
Total Population - Both sexes	251	Zambia	2018	17351708
Total Population - Both sexes	181	Zimbabwe	2018	14438802

237 rows × 4 columns

In [ ]:

```
df9 = df8.set_index("Area Code (FAO)")
df9
```

	Area	Year	value
Area Code (FAO)			
2	Afghanistan	2018	37171921
3	Albania	2018	2882740
4	Algeria	2018	42228408
5	American Samoa	2018	55465
6	Andorra	2018	77006
...	...	...	...
243	Wallis and Futuna Islands	2018	11661
205	Western Sahara	2018	567402
249	Yemen	2018	28498683
251	Zambia	2018	17351708
181	Zimbabwe	2018	14438802

237 rows × 3 columns

In [ ]:

```
df10 = pd.DataFrame(df9, index=[10,16,351,68,79,100,105,231,110,138,165,194,223])
df10
```

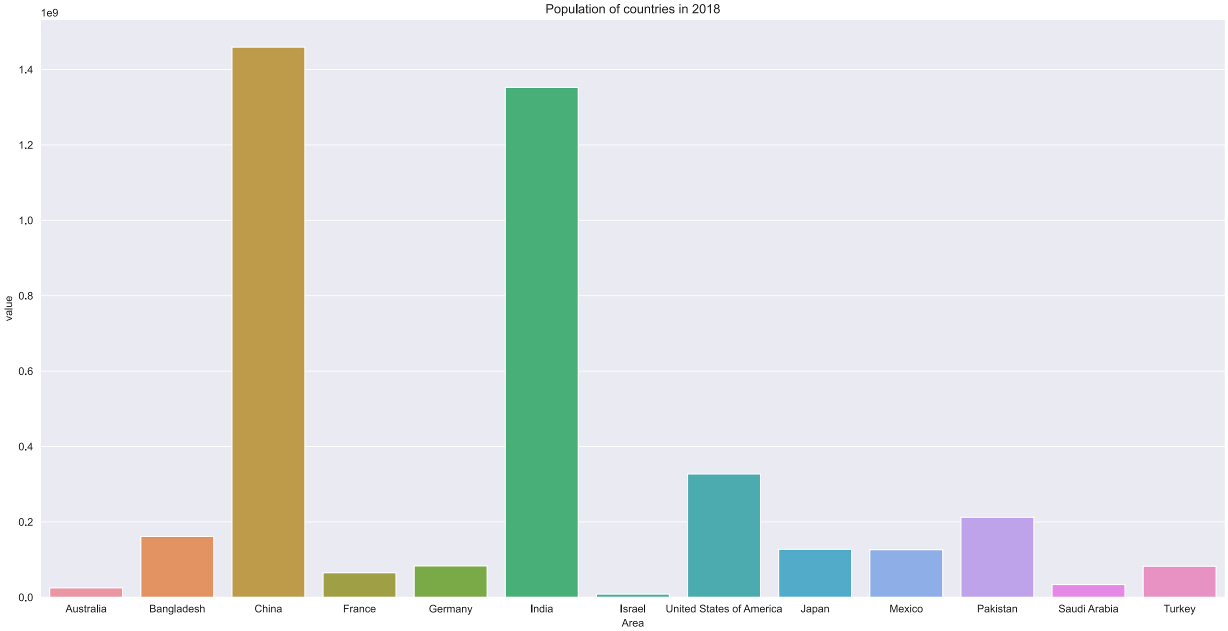
	Area	Year	value
10	Australia	2018	24898152
16	Bangladesh	2018	161376708
351	China	2018	1459377612
68	France	2018	64990511
79	Germany	2018	83124418
100	India	2018	1352642280
105	Israel	2018	8381516
231	United States of America	2018	327096265
110	Japan	2018	127202192
138	Mexico	2018	126190788
165	Pakistan	2018	212228286
194	Saudi Arabia	2018	33702756
223	Turkey	2018	82340088

In [ ]:

```
sns.set_style("darkgrid")
```

```
plt.figure(figsize=(20,10))
plot2 = sns.barplot(data=df10, x="Area", y="value")
plt.title("Population of countries in 2018")
plot2
```

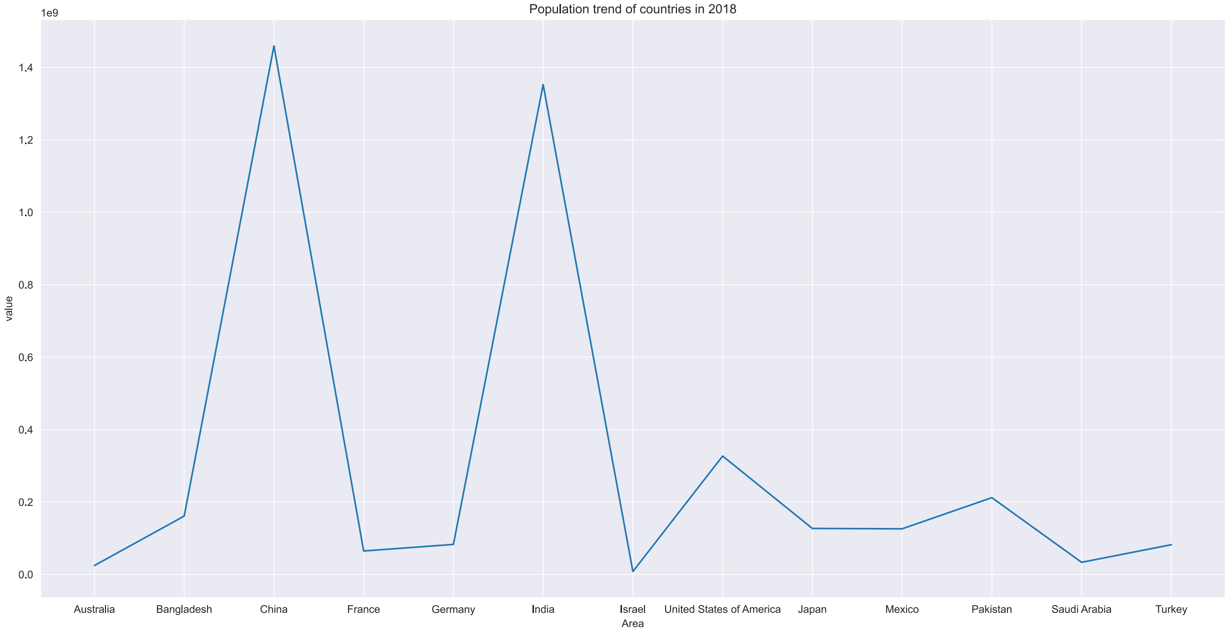
```
<AxesSubplot:title={'center':'Population of countries in 2018'}, xlabel='Area', ylabe
l='value'>
```



In [ ]:
```
sns.set_style("darkgrid")

plt.figure(figsize=(20,10))
plot2 = sns.lineplot(data=df10, x="Area", y="value")
plt.title("Population trend of countries in 2018")
plot2
```

```
<AxesSubplot:title={'center':'Population trend of countries in 2018'}, xlabel='Area',
ylabel='value'>
```



Exploratory Data Analysis

In [ ]:

```
#importing libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [ ]:

```
kashti= sns.load_dataset("titanic")
```

In [ ]:

```
kashti.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

In [ ]:

```
ks= kashti
```

In [ ]:

```
#just to see the datset
ks.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN

In [ ]:

```
ks.shape
#Rows x column
```

```
(891, 15)
```

In [ ]:

```
ks.tail()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	de
886	0	2	male	27.0	0	0	13.00	S	Second	man	True	Na
887	1	1	female	19.0	0	0	30.00	S	First	woman	False	
888	0	3	female	NaN	1	2	23.45	S	Third	woman	False	Na
889	1	1	male	26.0	0	0	30.00	C	First	man	True	
890	0	3	male	32.0	0	0	7.75	Q	Third	man	True	Na

In [ ]:

```
ks.describe()
```

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [ ]:

```
#  unique values
ks.nunique()
```

```
survived         2
pclass           3
sex              2
age             88
sibsp            7
parch            7
fare           248
embarked         3
class            3
who              3
adult_male       2
deck             7
embark_town      3
alive            2
alone            2
dtype: int64
```

In [ ]:

```
# coloumn names
ks.columns
```

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'deck', 'embark_town',
       'alive', 'alone'],
      dtype='object')
```

In [ ]:
```
ks["sex"].unique()
```

```
array(['male', 'female'], dtype=object)
```

In [ ]:
```
ks['who'].unique()
```

```
array(['man', 'woman', 'child'], dtype=object)
```

In [ ]:
```
np.union1d(ks["who"].unique(), ks["sex"].unique())
```

```
array(['child', 'female', 'male', 'man', 'woman'], dtype=object)
```

In [ ]:
```
ks[['who', "sex"]].nunique()
```

```
who    3
sex    2
dtype: int64
```

cleaning and filtering the data

In [ ]:
```
# find missing valus inside
ks.isnull()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	en
0	False	False	False	False	False	False	False	False	False	False	False	True	
1	False	False	False	False	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	False	False	False	True	
3	False	False	False	False	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	False	False	False	True	
...	...	...	...	...	...	...	...	...	...	...	...	...	
886	False	False	False	False	False	False	False	False	False	False	False	True	
887	False	False	False	False	False	False	False	False	False	False	False	False	
888	False	False	False	True	False	False	False	False	False	False	False	True	
889	False	False	False	False	False	False	False	False	False	False	False	False	
890	False	False	False	False	False	False	False	False	False	False	False	True	

891 rows × 15 columns

In [ ]:
```
ks.isnull().sum()
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

In [ ]:

```
# removing missing value column
ks_clean= ks.drop (["deck"], axis= 1)
ks_clean.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	emba
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Soutl
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	Cl
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Soutl
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	Soutl
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Soutl

In [ ]:

```
ks_clean.isnull().sum()
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
embark_town      2
alive            0
alone            0
dtype: int64
```

In [ ]:

```
ks_clean.shape
```

```
(891, 14)
```

In [ ]:

```
891-177-2
#117 row in age and 2 from embarked and embark town
```

```
712
```

```
In [ ]: ks_clean= ks_clean.dropna()
        ks_clean.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	emba
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Sout
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Sout
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	Sout
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Sout

```
In [ ]: ks_clean.dropna().shape
```

```
(712, 14)
```

```
In [ ]: ks_clean.isnull().sum()
```

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

```
In [ ]: ks_clean.shape
```

```
(712, 14)
```

```
In [ ]: ks.shape
```

```
(891, 15)
```

```
In [ ]: ks_clean["age"].value_counts()
```

```
24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
         ..
36.50     1
55.50     1
0.92      1
23.50     1
74.00     1
Name: age, Length: 88, dtype: int64
```

In [ ]:

```
ks_clean["sex"].value_counts()
```

```
male      453
female    259
Name: sex, dtype: int64
```

In [ ]:

```
ks_clean["sex"].value_counts()
```

```
male      453
female    259
Name: sex, dtype: int64
```

In [ ]:

```
ks.describe()
```

	survived	pclass	age	sibsp	parch	fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [ ]:

```
ks_clean.describe()
```

	survived	pclass	age	sibsp	parch	fare
count	712.000000	712.000000	712.000000	712.000000	712.000000	712.000000
mean	0.404494	2.240169	29.642093	0.514045	0.432584	34.567251
std	0.491139	0.836854	14.492933	0.930692	0.854181	52.938648
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	1.000000	20.000000	0.000000	0.000000	8.050000
50%	0.000000	2.000000	28.000000	0.000000	0.000000	15.645850
75%	1.000000	3.000000	38.000000	1.000000	1.000000	33.000000
max	1.000000	3.000000	80.000000	5.000000	6.000000	512.329200

In [ ]:

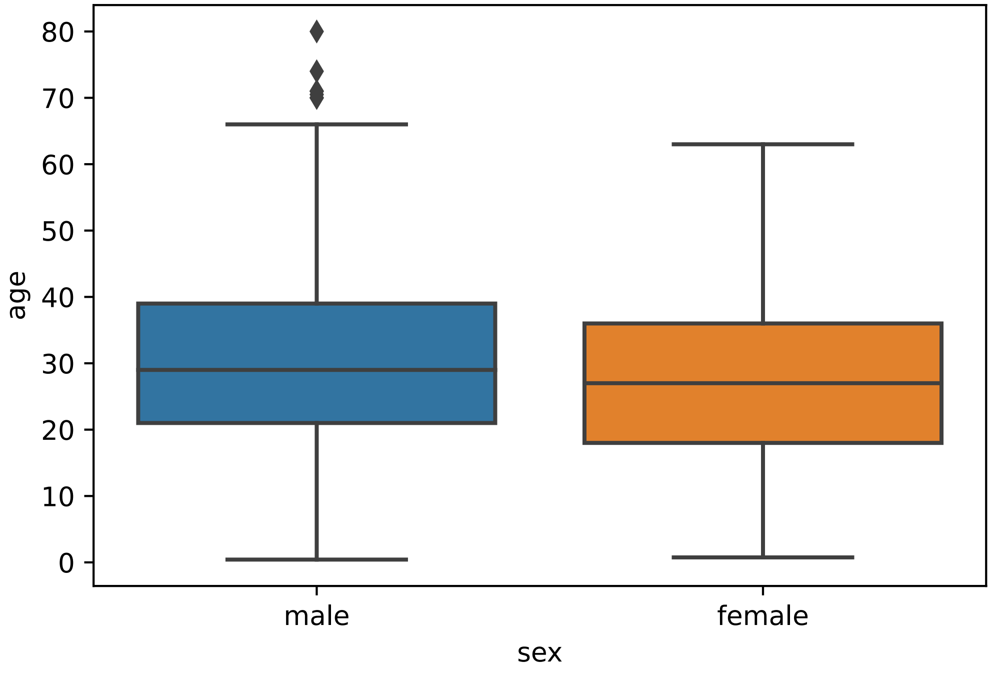
```
ks_clean.columns
```

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'embark_town', 'alive',
       'alone'],
      dtype='object')
```

In [ ]:

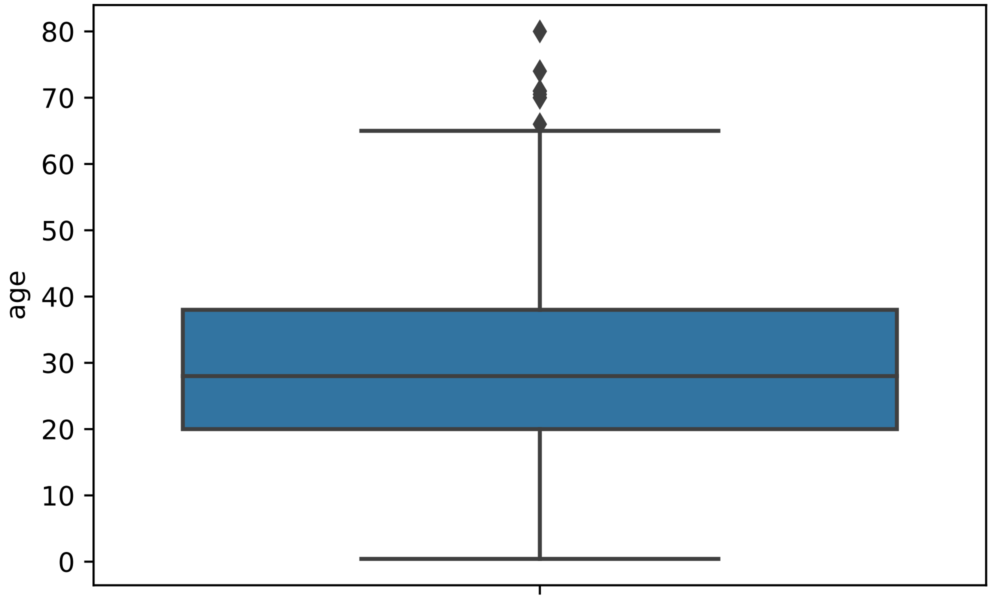
```
sns.boxplot(x= "sex", y='age', data =ks_clean )
# here we can see the outlyers in age
```

```
<AxesSubplot:xlabel='sex', ylabel='age'>
```



```
In [ ]: sns.boxplot(y='age', data =ks_clean )
```

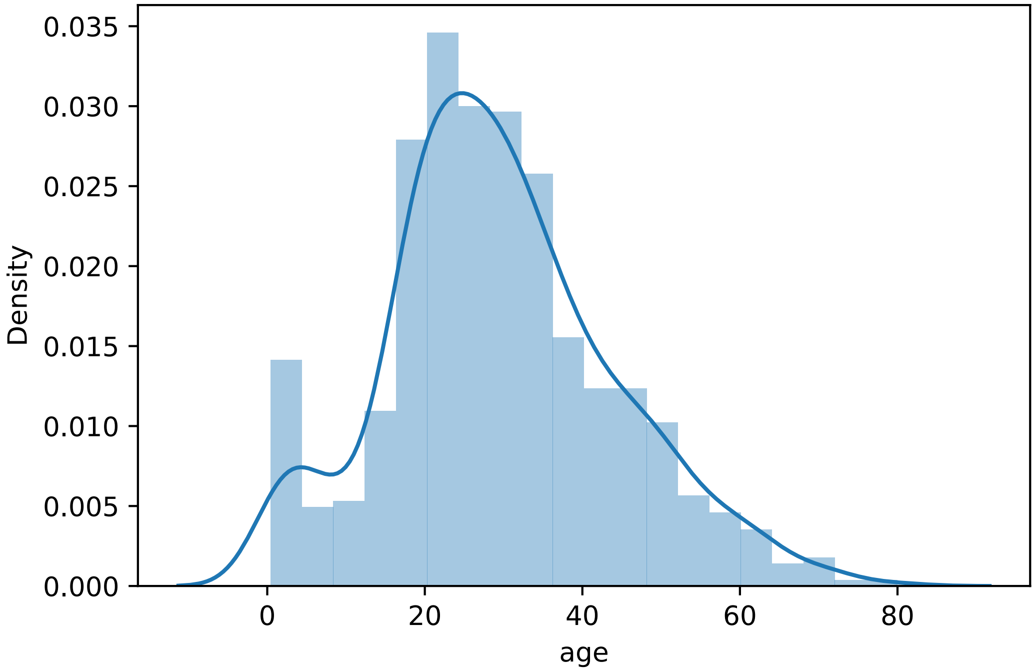
```
<AxesSubplot:ylabel='age'>
```



```
In [ ]: sns.distplot(ks_clean["age"] )

        # here we are seeing the bell curve / histogram for normality check
        # here we can see that it is not perfectly bell curve means data is not perfect
```

```
C:\Users\Arsl\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Pleas
e adapt your code to use either `displot` (a figure-level function with similar flexi
bility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
<AxesSubplot:xlabel='age', ylabel='Density'>
```



```
In [ ]: ks_clean["age"].mean()
```

```
29.64209269662921
```

```
In [ ]: ks_clean.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	emba
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Sout
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Sout
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	Sout
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Sout

```
In [ ]: # Removing an out liers
        ks_clean= ks_clean[ks_clean["age"]< 68]
        ks_clean.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	emba
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Sout
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Sout
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	Sout
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Sout

```
In [ ]: ks_clean.shape
```

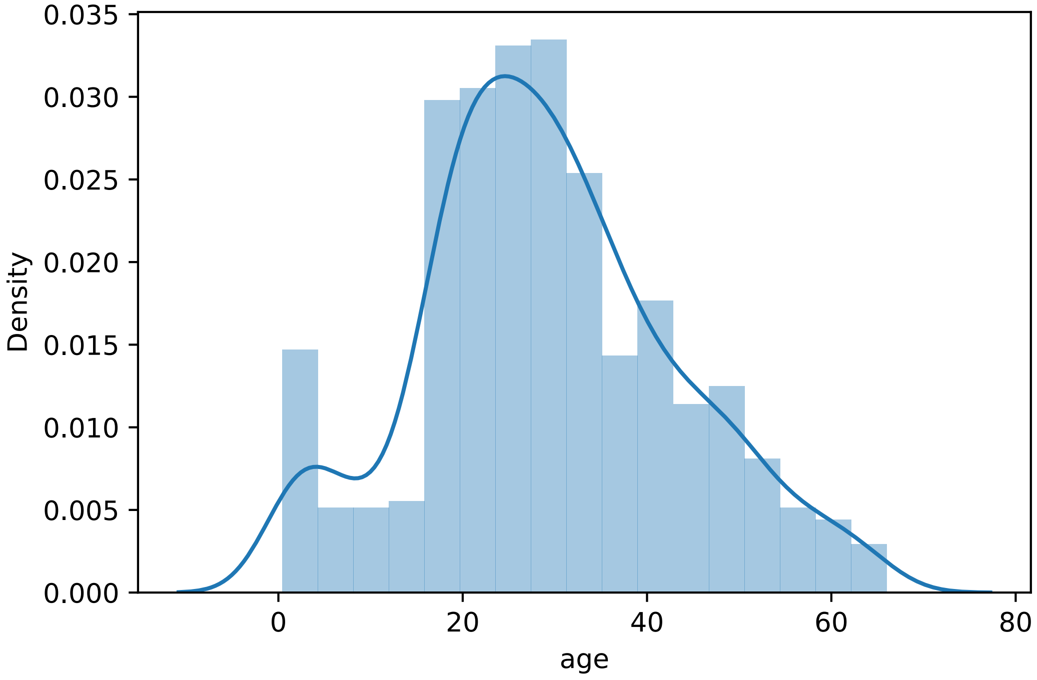
```
(705, 14)
```

```
In [ ]: ks_clean["age"].mean()
```

```
29.21797163120567
```

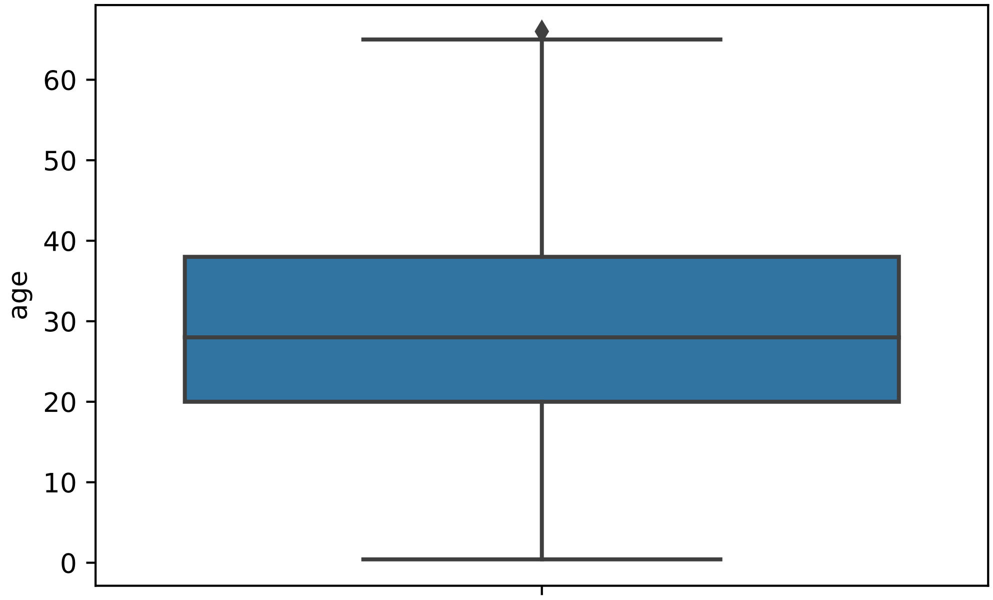
```
In [ ]: sns.distplot(ks_clean["age"] )
```

```
C:\Users\Arsl\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Pleas
e adapt your code to use either `displot` (a figure-level function with similar flexi
bility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
<AxesSubplot:xlabel='age', ylabel='Density'>
```



```
In [ ]: sns.boxplot(y='age', data =ks_clean )
```

```
<AxesSubplot:ylabel='age'>
```

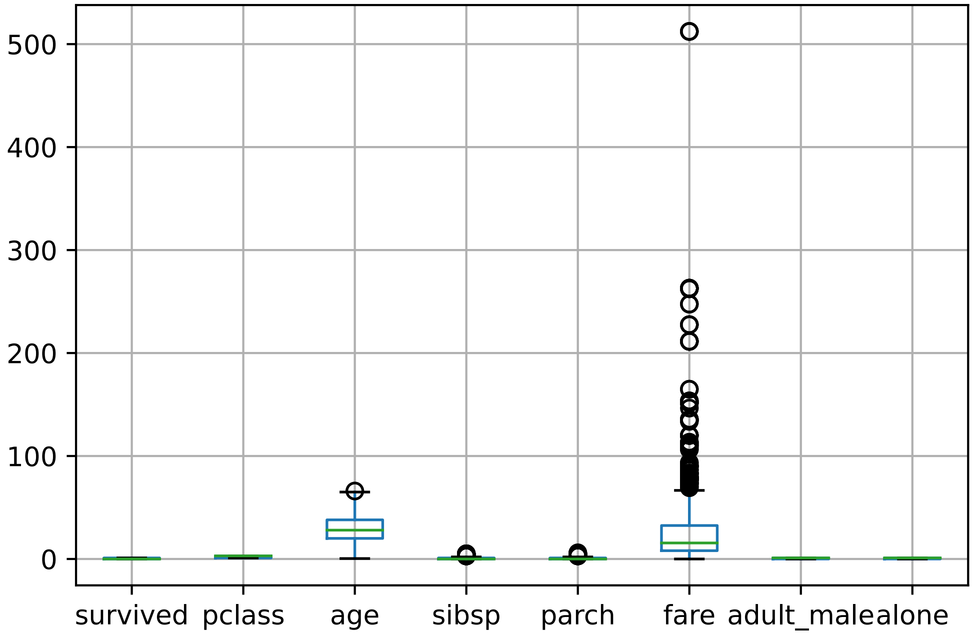


In [ ]:
```
ks_clean.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	emba
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Sout
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Sout
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	Sout
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Sout

In [ ]:
```
ks_clean.boxplot()
```

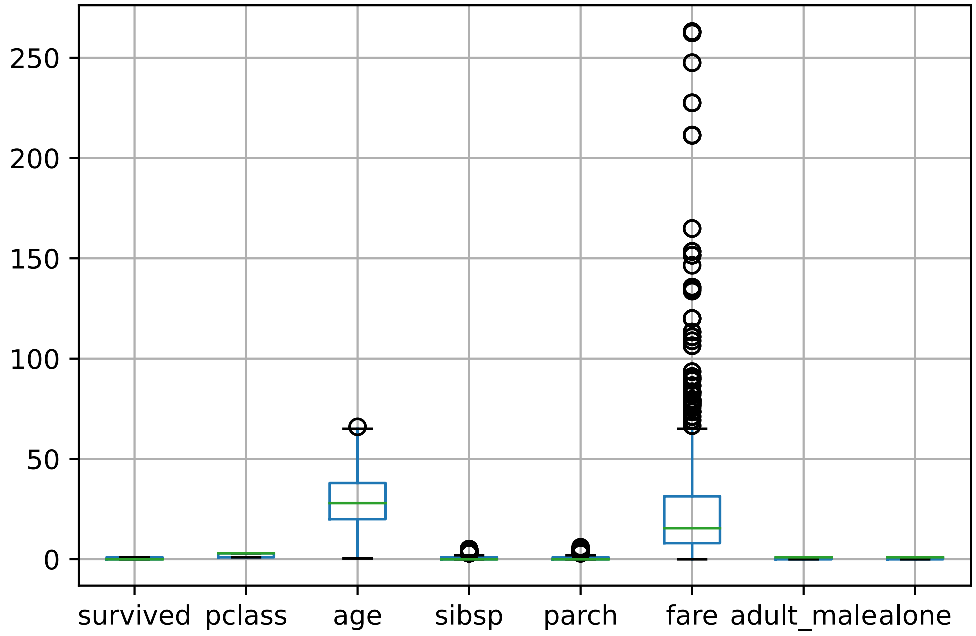
```
<AxesSubplot:>
```



In [ ]:
```
ks_clean= ks_clean[ks_clean["fare"]< 300]
```

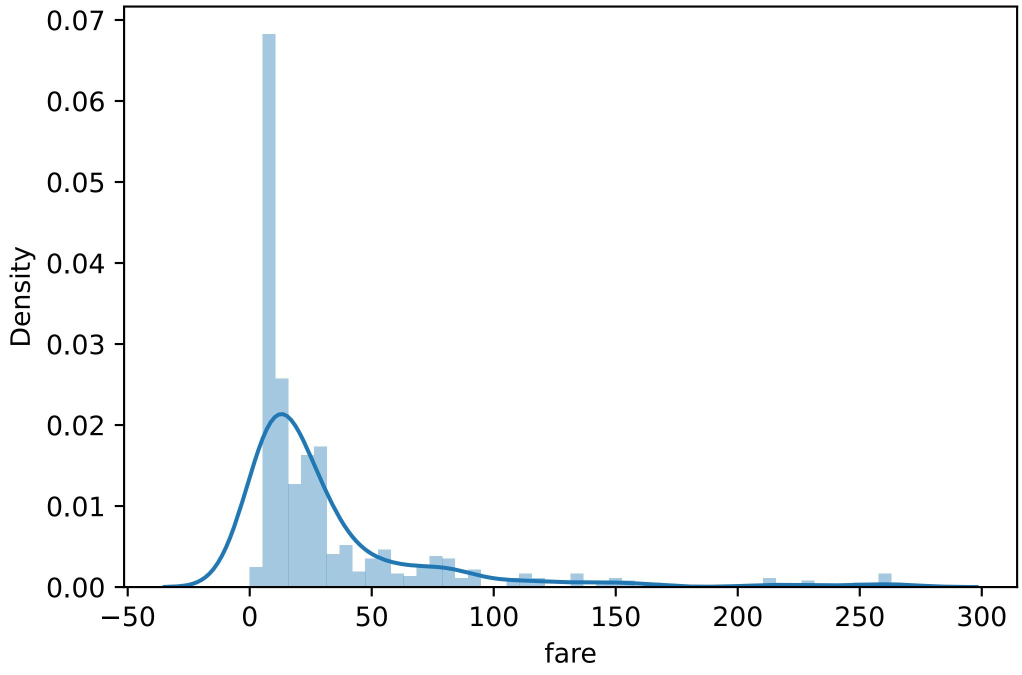
In [ ]:
```
ks_clean.boxplot()
```

```
<AxesSubplot:>
```



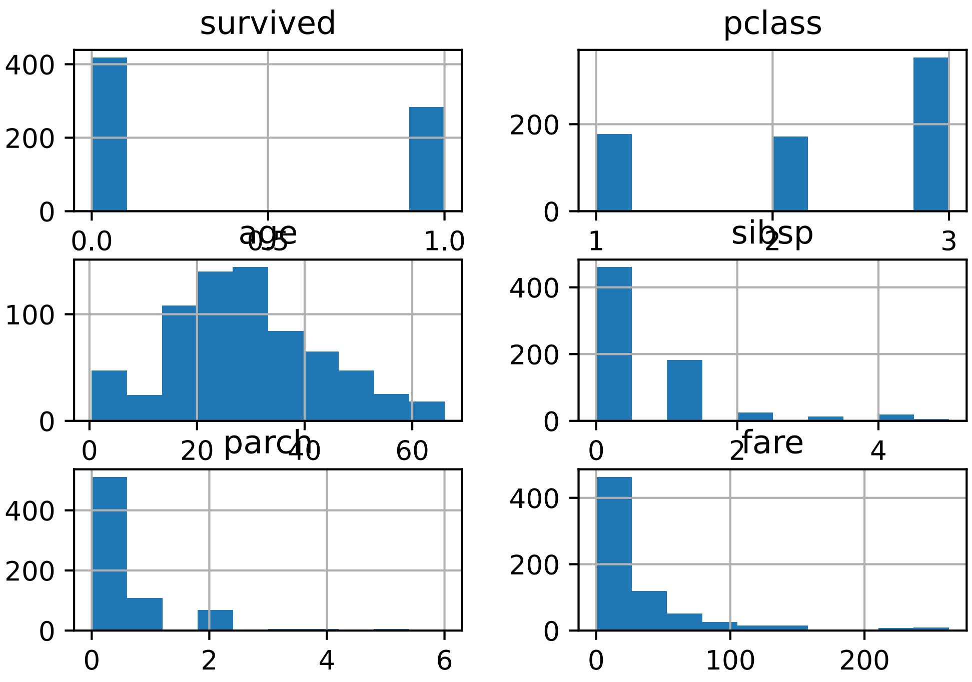
```
In [ ]: sns.distplot(ks_clean["fare"] )
```

```
C:\Users\Arsl\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarnin
g: `distplot` is a deprecated function and will be removed in a future version. Pleas
e adapt your code to use either `displot` (a figure-level function with similar flexi
bility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
<AxesSubplot:xlabel='fare', ylabel='Density'>
```



```
In [ ]: ks_clean.hist()
```

```
array([[<AxesSubplot:title={'center':'survived'}>,
        <AxesSubplot:title={'center':'pclass'}>],
       [<AxesSubplot:title={'center':'age'}>,
        <AxesSubplot:title={'center':'sibsp'}>],
       [<AxesSubplot:title={'center':'parch'}>,
        <AxesSubplot:title={'center':'fare'}>]], dtype=object)
```

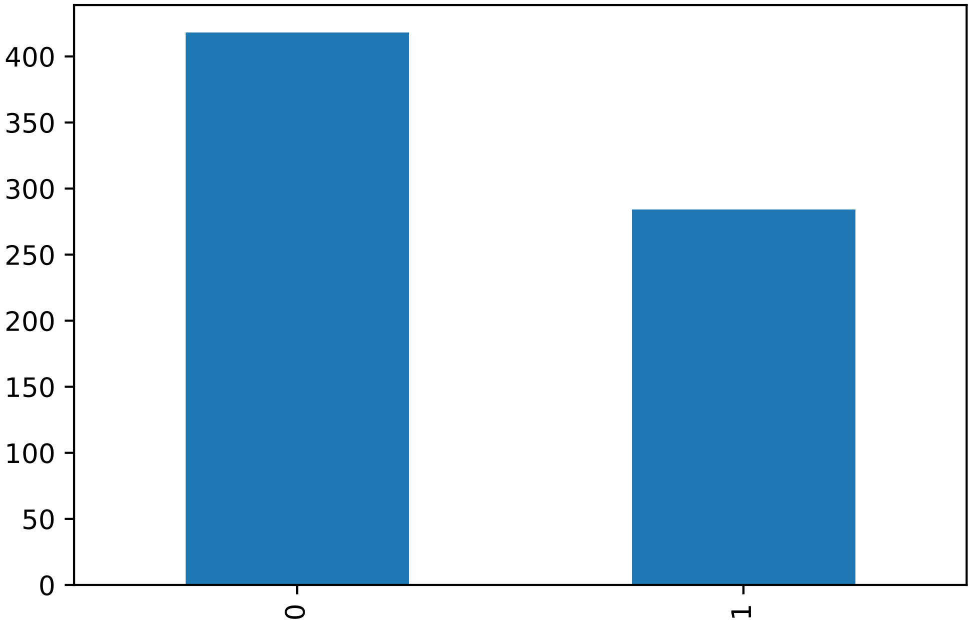


In [ ]:
```
pd.value_counts(ks_clean["survived"])
```

```
0    418
1    284
Name: survived, dtype: int64
```

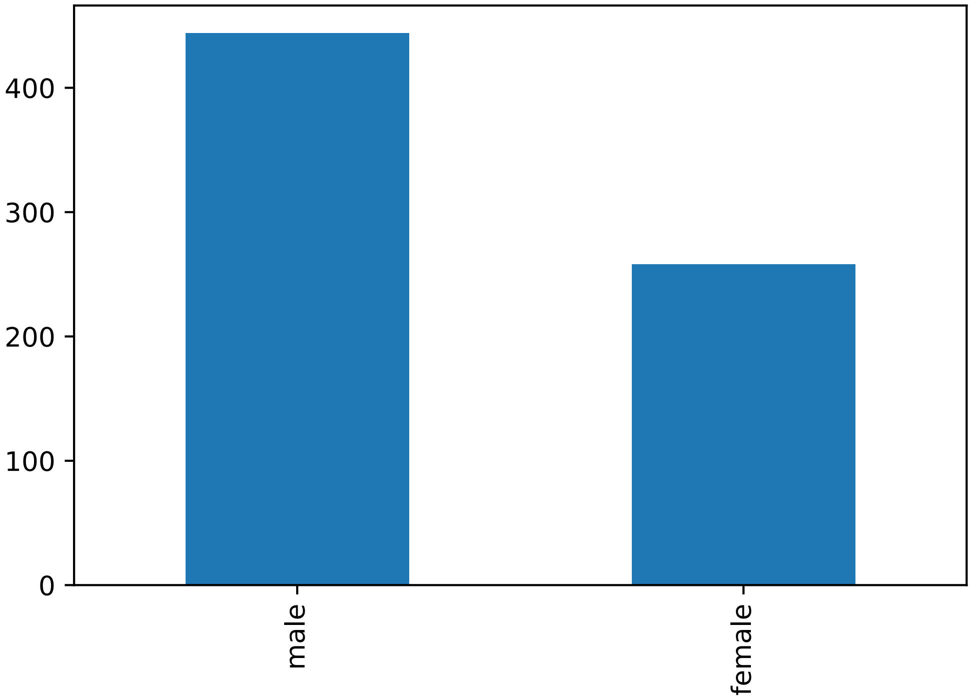
In [ ]:
```
pd.value_counts(ks_clean["survived"]).plot.bar()
```

```
<AxesSubplot:>
```



In [ ]:
```
pd.value_counts(ks_clean["sex"]).plot.bar()
```

```
<AxesSubplot:>
```



In [ ]:

```
ks_clean.groupby(["sex","class","who"]).mean()
```

sex	class	who	survived	pclass	age	sibsp	parch	fare	adult_male	
female	First	child	0.666667	1.0	10.333333	0.666667	1.666667	160.962500	0.0	0.0
		man	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
		woman	0.974684	1.0	35.139241	0.556962	0.468354	101.521730	0.0	0.3
	Second	child	1.000000	2.0	6.600000	0.700000	1.300000	29.240000	0.0	0.0
		man	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
		woman	0.906250	2.0	32.179688	0.468750	0.515625	20.812175	0.0	0.4
	Third	child	0.533333	3.0	7.100000	1.533333	1.100000	19.023753	0.0	0.1
		man	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
		woman	0.430556	3.0	27.854167	0.527778	0.888889	14.563542	0.0	0.4
male	First	child	1.000000	1.0	5.306667	0.666667	2.000000	117.802767	0.0	0.0
		man	0.369565	1.0	41.201087	0.380435	0.282609	61.110824	1.0	0.5
		woman	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
	Second	child	1.000000	2.0	2.258889	0.888889	1.222222	27.306022	0.0	0.0
		man	0.067416	2.0	33.179775	0.325843	0.146067	20.606133	1.0	0.6
		woman	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
	Third	child	0.321429	3.0	6.515000	2.821429	1.321429	27.716371	0.0	0.0
		man	0.130045	3.0	28.607623	0.201794	0.125561	10.249231	1.0	0.8
		woman	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

In [ ]:

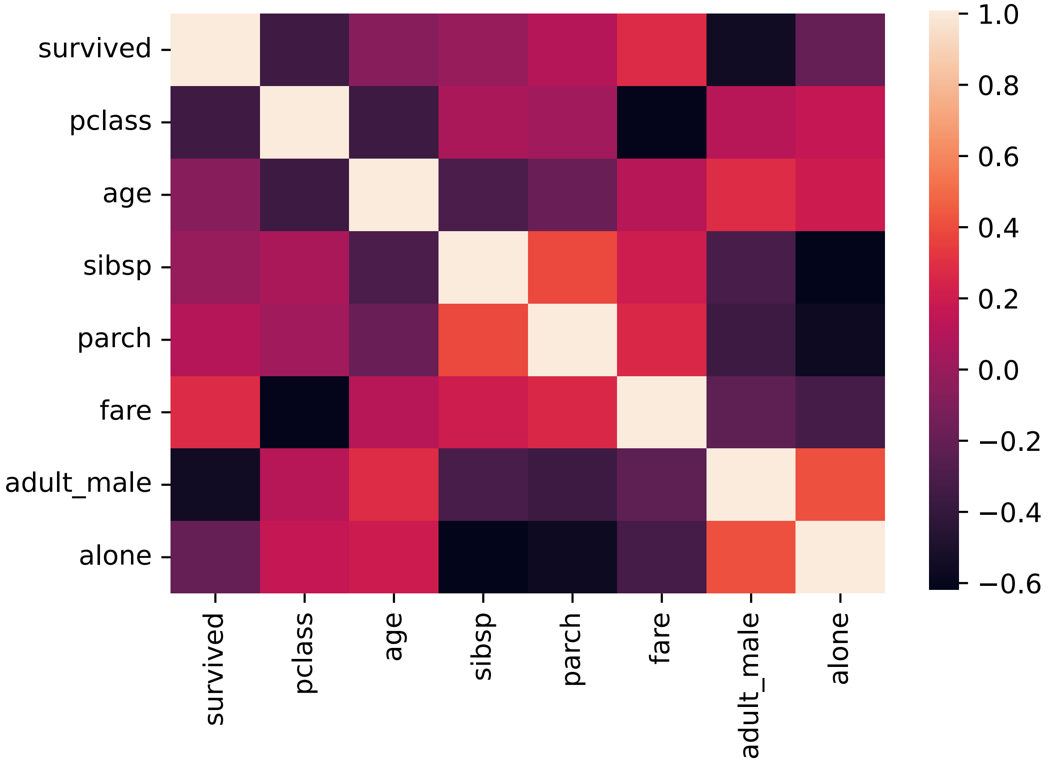
```
cor_ks_clean= ks_clean.corr()
cor_ks_clean
```

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.356549	-0.074335	-0.014483	0.095426	0.273531	-0.554567	-0.201175
pclass	-0.356549	1.000000	-0.365121	0.061354	0.022519	-0.617591	0.102930	0.156030
age	-0.074335	-0.365121	1.000000	-0.308906	-0.186271	0.103100	0.275035	0.187284
sibsp	-0.014483	0.061354	-0.308906	1.000000	0.381803	0.197954	-0.311622	-0.629200
parch	0.095426	0.022519	-0.186271	0.381803	1.000000	0.259948	-0.366540	-0.574701
fare	0.273531	-0.617591	0.103100	0.197954	0.259948	1.000000	-0.228675	-0.333949
adult_male	-0.554567	0.102930	0.275035	-0.311622	-0.366540	-0.228675	1.000000	0.402214
alone	-0.201175	0.156030	0.187284	-0.629200	-0.574701	-0.333949	0.402214	1.000000

In [ ]:

```
sns.heatmap(cor_ks_clean)
```

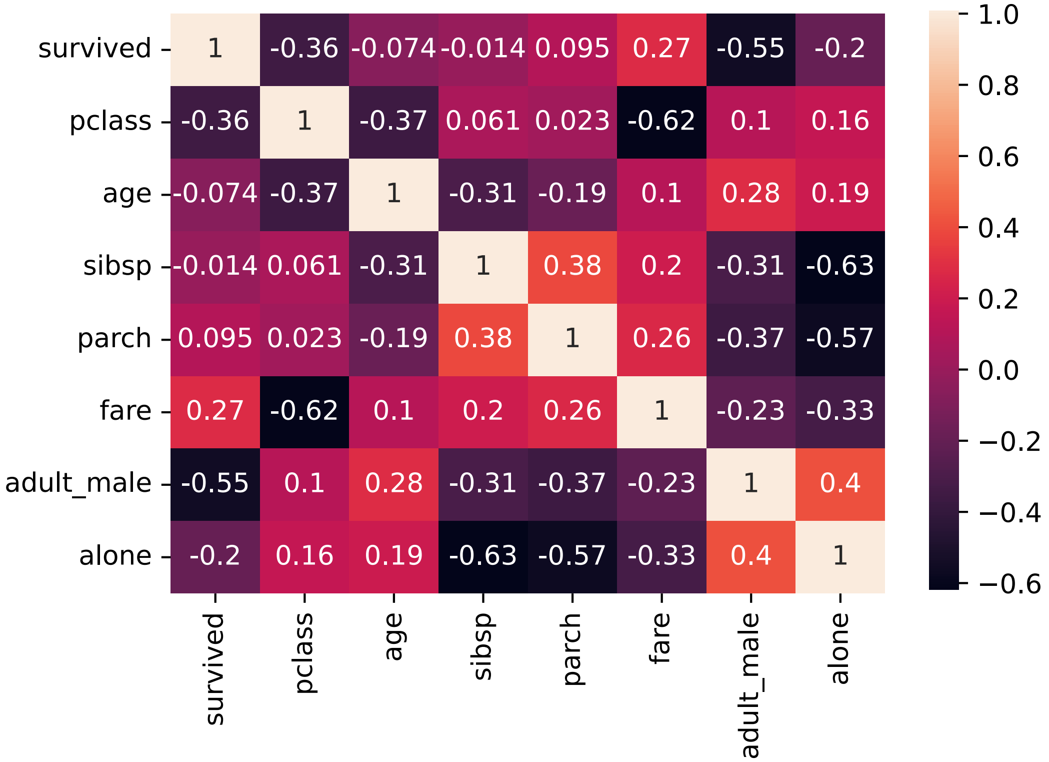
```
<AxesSubplot:>
```



In [ ]:

```
sns.heatmap(cor_ks_clean , annot=True)
```

```
<AxesSubplot:>
```



In [ ]:

```
sns.relplot(x= "age", y="fare", hue="sex", data=ks_clean)
```

```
<seaborn.axisgrid.FacetGrid at 0x1d0b5f83700>
```