Python Chilla Pandas Assignment

> **Title= "Mr"\ Name= "Ali Nawaz"\ email = "nawazktk99@gmail.com"\ whatsapp = "03358043653"\ Artificial Intelligence Engineer at NUST\ Education : Master in Software Engineering**

Pandas Detail Hands On Experience with Baba Ammar

In this notebook we are going to handon experience on `pandas` library and doing diffrent operation on `csv` data. Let's start

In [ ]:
```
# pip install pandas
```

In [ ]:
```
import plotly.express as px
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
# object creation
data = pd.Series(['name', 'contact', 'subject', 'address', 'age'])# to ad empty cel or
data
```

Out[ ]:
```
0       name
1    contact
2    subject
3    address
4        age
dtype: object
```

In [ ]:
```
# we can do date range like below
dates = pd.date_range('20220101', periods=10)#formate can be diffrent as per your style
dates
```

Out[ ]:
```
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'],
              dtype='datetime64[ns]', freq='D')
```

In [ ]:
```
df = pd.DataFrame(np.random.randn(10,4), index=dates, columns=list('ABCD'))
df
```

Out[ ]:

	A	B	C	D
2022-01-01	1.483728	0.965460	0.786022	0.923444
2022-01-02	0.917465	0.076732	1.095824	0.887206

	A	B	C	D
2022-01-03	-0.662774	0.793088	-1.863862	0.329197
2022-01-04	-3.458971	1.589321	-0.360993	-3.635660
2022-01-05	0.889227	-0.136649	0.811274	0.253537
2022-01-06	0.592051	0.949553	1.148953	0.348969
2022-01-07	-0.082928	-0.398239	-0.039909	-1.040645
2022-01-08	-1.325646	-0.861409	-1.282401	1.144879
2022-01-09	0.645953	-0.999863	-1.128131	-0.395491
2022-01-10	1.388839	1.579899	-1.052474	0.014124

In [ ]:
```
df2 = pd.DataFrame(
    {
        "A" : 1.0,
        "B" : pd.Timestamp('20220101'),
        "C" : pd.Series(1, index = list(range(4)), dtype= 'float32'),
        "D" : np.array([3]* 4, dtype= 'int32'),
        "E" : pd.Categorical(["boys", "man","boys", "man"]),
        "F" : "mans"
    }
)
df2.head()
```

Out[ ]:

	A	B	C	D	E	F
0	1.0	2022-01-01	1.0	3	boys	mans
1	1.0	2022-01-01	1.0	3	man	mans
2	1.0	2022-01-01	1.0	3	boys	mans
3	1.0	2022-01-01	1.0	3	man	mans

In [ ]:
```
df2.index
```

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

In [ ]:
```
arr = df.to_numpy()
arr
```

Out[ ]:
```
array([[ 1.48372788,  0.96546006,  0.78602153,  0.92344353],
       [ 0.91746526,  0.0767318 ,  1.09582422,  0.88720637],
       [-0.66277402,  0.79308791, -1.86386226,  0.32919745],
       [-3.45897133,  1.58932105, -0.36099349, -3.63566046],
       [ 0.88922692, -0.13664891,  0.8112742 ,  0.25353715],
       [ 0.59205137,  0.94955282,  1.14895257,  0.3489694 ],
       [-0.08292772, -0.39823876, -0.0399094 , -1.04064504],
       [-1.32564554, -0.86140929, -1.28240138,  1.14487916],
       [ 0.64595335, -0.99986252, -1.12813141, -0.39549078],
       [ 1.38883918,  1.57989854, -1.05247436,  0.01412359]])
```

In [ ]:
```
df.describe()
```

Out[ ]:

	A	B	C	D
count	10.000000	10.000000	10.000000	10.000000
mean	0.038695	0.355789	-0.188570	-0.117044
std	1.515973	0.951246	1.109162	1.397047
min	-3.458971	-0.999863	-1.863862	-3.635660
25%	-0.517812	-0.332841	-1.109217	-0.293087
50%	0.619002	0.434910	-0.200451	0.291367
75%	0.910406	0.961483	0.804961	0.752647
max	1.483728	1.589321	1.148953	1.144879

> **To Transpose DataFrame**

In [ ]:
```
# To traspose Data
df2.T
```

Out[ ]:

	0	1	2	3
A	1	1	1	1
B	2022-01-01 00:00:00	2022-01-01 00:00:00	2022-01-01 00:00:00	2022-01-01 00:00:00
C	1	1	1	1
D	3	3	3	3
E	boys	man	boys	man
F	mans	mans	mans	mans

In [ ]:
```
a = df.sort_index(axis=1, ascending=True)
a
```

Out[ ]:

	A	B	C	D
2022-01-01	1.483728	0.965460	0.786022	0.923444
2022-01-02	0.917465	0.076732	1.095824	0.887206
2022-01-03	-0.662774	0.793088	-1.863862	0.329197
2022-01-04	-3.458971	1.589321	-0.360993	-3.635660
2022-01-05	0.889227	-0.136649	0.811274	0.253537
2022-01-06	0.592051	0.949553	1.148953	0.348969
2022-01-07	-0.082928	-0.398239	-0.039909	-1.040645
2022-01-08	-1.325646	-0.861409	-1.282401	1.144879

	A	B	C	D
2022-01-09	0.645953	-0.999863	-1.128131	-0.395491
2022-01-10	1.388839	1.579899	-1.052474	0.014124

In [ ]:
```
b = df.sort_values(by='C', ascending=False)
b
```

Out[ ]:

	A	B	C	D
2022-01-06	0.592051	0.949553	1.148953	0.348969
2022-01-02	0.917465	0.076732	1.095824	0.887206
2022-01-05	0.889227	-0.136649	0.811274	0.253537
2022-01-01	1.483728	0.965460	0.786022	0.923444
2022-01-07	-0.082928	-0.398239	-0.039909	-1.040645
2022-01-04	-3.458971	1.589321	-0.360993	-3.635660
2022-01-10	1.388839	1.579899	-1.052474	0.014124
2022-01-09	0.645953	-0.999863	-1.128131	-0.395491
2022-01-08	-1.325646	-0.861409	-1.282401	1.144879
2022-01-03	-0.662774	0.793088	-1.863862	0.329197

In [ ]:
```
df.A
df['A']
```

Out[ ]:
```
2022-01-01    1.483728
2022-01-02    0.917465
2022-01-03   -0.662774
2022-01-04   -3.458971
2022-01-05    0.889227
2022-01-06    0.592051
2022-01-07   -0.082928
2022-01-08   -1.325646
2022-01-09    0.645953
2022-01-10    1.388839
Freq: D, Name: A, dtype: float64
```

> Row wise Selection

In [ ]:
```
df[0:2]
df[0:1]
df[0:5]
df[0:]
```

Out[ ]:

	A	B	C	D
2022-01-01	1.483728	0.965460	0.786022	0.923444

	A	B	C	D
2022-01-02	0.917465	0.076732	1.095824	0.887206
2022-01-03	-0.662774	0.793088	-1.863862	0.329197
2022-01-04	-3.458971	1.589321	-0.360993	-3.635660
2022-01-05	0.889227	-0.136649	0.811274	0.253537
2022-01-06	0.592051	0.949553	1.148953	0.348969
2022-01-07	-0.082928	-0.398239	-0.039909	-1.040645
2022-01-08	-1.325646	-0.861409	-1.282401	1.144879
2022-01-09	0.645953	-0.999863	-1.128131	-0.395491
2022-01-10	1.388839	1.579899	-1.052474	0.014124

Select by Labels loc

In [ ]:
```
df.loc[dates[5]]
```

Out[ ]:
```
A    0.592051
B    0.949553
C    1.148953
D    0.348969
Name: 2022-01-06 00:00:00, dtype: float64
```

In [ ]:
```
# select columns and rows
df.loc[:, ['A', 'B']] # geting few but all rows of dataa
```

Out[ ]:

	A	B
2022-01-01	1.483728	0.965460
2022-01-02	0.917465	0.076732
2022-01-03	-0.662774	0.793088
2022-01-04	-3.458971	1.589321
2022-01-05	0.889227	-0.136649
2022-01-06	0.592051	0.949553
2022-01-07	-0.082928	-0.398239
2022-01-08	-1.325646	-0.861409
2022-01-09	0.645953	-0.999863
2022-01-10	1.388839	1.579899

In [ ]:
```
# getting sliced data of that columns

df.loc['20220105':'20220115', ['A', 'B']]
# df.loc[['20220105','20220110'], ['A', 'B']] # specific values based rows extraction
```

Out[ ]:

	A	B
2022-01-05	0.889227	-0.136649
2022-01-06	0.592051	0.949553
2022-01-07	-0.082928	-0.398239
2022-01-08	-1.325646	-0.861409
2022-01-09	0.645953	-0.999863
2022-01-10	1.388839	1.579899

In [ ]:
```
df.loc['20220104', ['A', 'B', 'C']]
```

Out[ ]:
```
A   -3.458971
B    1.589321
C   -0.360993
Name: 2022-01-04 00:00:00, dtype: float64
```

In [ ]:
```
# get single datacell
df.at[dates[4], 'A']
```

Out[ ]:
```
0.8892269245998489
```

In [ ]:
```
df.iloc[0:5]
```

Out[ ]:

	A	B	C	D
2022-01-01	1.483728	0.965460	0.786022	0.923444
2022-01-02	0.917465	0.076732	1.095824	0.887206
2022-01-03	-0.662774	0.793088	-1.863862	0.329197
2022-01-04	-3.458971	1.589321	-0.360993	-3.635660
2022-01-05	0.889227	-0.136649	0.811274	0.253537

In [ ]:
```
df.iloc[0:5, 0:2] #first is ros and 2nd is columns
```

Out[ ]:

	A	B
2022-01-01	1.483728	0.965460
2022-01-02	0.917465	0.076732
2022-01-03	-0.662774	0.793088
2022-01-04	-3.458971	1.589321
2022-01-05	0.889227	-0.136649

In [ ]:
```
df.iloc[0:5, :]
```

Out[ ]:

	A	B	C	D
2022-01-01	1.483728	0.965460	0.786022	0.923444
2022-01-02	0.917465	0.076732	1.095824	0.887206
2022-01-03	-0.662774	0.793088	-1.863862	0.329197
2022-01-04	-3.458971	1.589321	-0.360993	-3.635660
2022-01-05	0.889227	-0.136649	0.811274	0.253537

In [ ]:
```
# colum ith string methodd
df.iloc[:, 0:3]
```

Out[ ]:

	A	B	C
2022-01-01	1.483728	0.965460	0.786022
2022-01-02	0.917465	0.076732	1.095824
2022-01-03	-0.662774	0.793088	-1.863862
2022-01-04	-3.458971	1.589321	-0.360993
2022-01-05	0.889227	-0.136649	0.811274
2022-01-06	0.592051	0.949553	1.148953
2022-01-07	-0.082928	-0.398239	-0.039909
2022-01-08	-1.325646	-0.861409	-1.282401
2022-01-09	0.645953	-0.999863	-1.128131
2022-01-10	1.388839	1.579899	-1.052474

In [ ]:
```
df[df['A']>0] #greater value in A print
```

Out[ ]:

	A	B	C	D
2022-01-01	1.483728	0.965460	0.786022	0.923444
2022-01-02	0.917465	0.076732	1.095824	0.887206
2022-01-05	0.889227	-0.136649	0.811274	0.253537
2022-01-06	0.592051	0.949553	1.148953	0.348969
2022-01-09	0.645953	-0.999863	-1.128131	-0.395491
2022-01-10	1.388839	1.579899	-1.052474	0.014124

In [ ]:
```
# on whole dataframe find the greater value than 0
df[df>0]
```

Out[ ]:

	A	B	C	D
2022-01-01	1.483728	0.965460	0.786022	0.923444

	A	B	C	D
2022-01-02	0.917465	0.076732	1.095824	0.887206
2022-01-03	NaN	0.793088	NaN	0.329197
2022-01-04	NaN	1.589321	NaN	NaN
2022-01-05	0.889227	NaN	0.811274	0.253537
2022-01-06	0.592051	0.949553	1.148953	0.348969
2022-01-07	NaN	NaN	NaN	NaN
2022-01-08	NaN	NaN	NaN	1.144879
2022-01-09	0.645953	NaN	NaN	NaN
2022-01-10	1.388839	1.579899	NaN	0.014124

In [ ]:
```
# copying dataframe like
df3 = df.copy()
```

In [ ]:
```
df.shape
```

Out[ ]:
```
(10, 4)
```

In [ ]:
```
# adding new col with exact no of values or rows as in orginal
df3['New']= [1,2,3,3,4,5, 1,2,3,3]
df3
```

Out[ ]:

	A	B	C	D	New
2022-01-01	1.483728	0.965460	0.786022	0.923444	1
2022-01-02	0.917465	0.076732	1.095824	0.887206	2
2022-01-03	-0.662774	0.793088	-1.863862	0.329197	3
2022-01-04	-3.458971	1.589321	-0.360993	-3.635660	3
2022-01-05	0.889227	-0.136649	0.811274	0.253537	4
2022-01-06	0.592051	0.949553	1.148953	0.348969	5
2022-01-07	-0.082928	-0.398239	-0.039909	-1.040645	1
2022-01-08	-1.325646	-0.861409	-1.282401	1.144879	2
2022-01-09	0.645953	-0.999863	-1.128131	-0.395491	3
2022-01-10	1.388839	1.579899	-1.052474	0.014124	3

In [ ]:
```

```

Numpy Handson Practice

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib as plt
import seaborn as sas
```

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

```
2.0
```

In [ ]:
```
data1 = [6,7,8.5,9]
#In python data1 is list, list can be accepted any datatype
import numpy as np
#lets convert the python list into Numpy array
arr1 = np.array(data1)
#If you print out the Numpy array list everything hasbeen changed to float, means numpy
print(arr1)
print(arr1.ndim)
```

```
[6.  7.  8.5 9. ]
1
```

In [ ]:
```
data2 = [[1,2,3,4],[5,6,7,8]]
arr2 = np.array(data2)
print(arr2)
print(arr2.shape)
print(arr2.ndim)

print(arr1.dtype)
print(arr2.dtype)
```

```
[[1 2 3 4]
 [5 6 7 8]]
(2, 4)
2
float64
int32
```

In [ ]:
```
np.empty((2,3,2))
```

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

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

In [ ]:
```
np.arange(15)  #Arange function
array1=np.arange(0,20,2,dtype=np.float64) # arange function, get an array in step with
print(array1)
```

```
[ 0.  2.  4.  6.  8. 10. 12. 14. 16. 18.]
```

In [ ]:
```
arr = np.array([1,2,3,4,5,6])#initialize array with int type
print(arr)
print(arr.dtype)
```

```
[1 2 3 4 5 6]
int32
```

In [ ]:
```
#calling astype always creating new arrays,, of same or different datatype
float_arr = arr.astype(np.float64)# convert the int array to float type with astype
float_arr
```

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

In [ ]:
```
#lets us cast flaot to int type
array_float = np.arange(0,100,3.87)
array_float
array_float.dtype
```

```
dtype('float64')
```

In [ ]:
```
array_int = array_float.astype(np.int32)
array_int
```

```
array([ 0,  3,  7, 11, 15, 19, 23, 27, 30, 34, 38, 42, 46, 50, 54, 58, 61,
       65, 69, 73, 77, 81, 85, 89, 92, 96])
```

In [ ]:
```
# so in above example flaot hasbeen converted to integer and decimal valuse has been tr
#lets convert the array of ints to array of strings
array_string = array_int.astype(np.str)
array_string
```

```
array(['0', '3', '7', '11', '15', '19', '23', '27', '30', '34', '38',
       '42', '46', '50', '54', '58', '61', '65', '69', '73', '77', '81',
       '85', '89', '92', '96'], dtype='<U11')
```

Array arithmatic operations

In [ ]:
```
x = np.arange(1,21,1).astype(np.float64)
x.shape = (4,5)
x
```

```
array([[ 1.,  2.,  3.,  4.,  5.],
       [ 6.,  7.,  8.,  9., 10.],
       [11., 12., 13., 14., 15.],
       [16., 17., 18., 19., 20.]])
```

In [ ]:
```
  #we can assign a matrix easily with reshape
y = np.arange(21,41,1).reshape(4,5)
y
```

```
array([[21, 22, 23, 24, 25],
       [26, 27, 28, 29, 30],
       [31, 32, 33, 34, 35],
       [36, 37, 38, 39, 40]])
```

In [ ]:
```
#addition
x+y
```

```
array([[22., 24., 26., 28., 30.],
```

```
       [32., 34., 36., 38., 40.],
       [42., 44., 46., 48., 50.],
       [52., 54., 56., 58., 60.]])
```

In [ ]:
```
#multiplication
x * y
```

```
array([[ 21.,  44.,  69.,  96., 125.],
       [156., 189., 224., 261., 300.],
       [341., 384., 429., 476., 525.],
       [576., 629., 684., 741., 800.]])
```

In [ ]:
```
x/0
0/x
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\ipykernel_launcher.py:1: Run
timeWarning: divide by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
array([[0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0.]])
```

In [ ]:
```
print(x**y)
print(x**2)
```

```
[[1.00000000e+00 4.19430400e+06 9.41431788e+10 2.81474977e+14
  2.98023224e+17]
 [1.70581728e+20 6.57123624e+22 1.93428131e+25 4.71012870e+27
  1.00000000e+30]
 [1.91943425e+32 3.41821892e+34 5.75613043e+36 9.29722225e+38
  1.45610961e+41]
 [2.23007452e+43 3.36209585e+45 5.01597304e+47 7.43687423e+49
  1.09951163e+52]]
[[  1.   4.   9.  16.  25.]
 [ 36.  49.  64.  81. 100.]
 [121. 144. 169. 196. 225.]
 [256. 289. 324. 361. 400.]]
```

In [ ]:
```
#basic indexing and slicing

x[3]
```

```
array([16., 17., 18., 19., 20.])
```

In [ ]:
```
x[1:3]
```

```
array([[ 6.,  7.,  8.,  9., 10.],
       [11., 12., 13., 14., 15.]])
```

In [ ]:
```
x[1:3,2:]
```

```
array([[ 8.,  9., 10.],
       [13., 14., 15.]])
```

In [ ]:
```
(x**2)[1:3,2:4] #logical
```

```
array([[ 64.,  81.],
       [169., 196.]])
```

In [ ]:
```
np.nan == np.nan
x[3,4]
(x**2)[2,3]
```

```
196.0
```

In [ ]:
```
# Arrays can be two-dimensional (e.g. to represent matrices):
a = np.ones([7,4])
print(a)

# an array has several attributes:
print(a.shape)
print(a.size)
print(a.dtype) # usually float64 (double), or int64 (long)
```

```
[[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.]]
(7, 4)
28
float64
```

In [ ]:
```
#multidimensional array
#3Dimensional
arr3d = np.array([[[1,2,3,],[4,5,6]],[[7,8,9],[10,11,12]]])
arr3d
arr3d[0]
```

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

In [ ]:
```
old_values = arr3d[0].copy()
old_values
```

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

In [ ]:
```
arr3d[0] = 50
arr3d[0]
```

```
array([[50, 50, 50],
       [50, 50, 50]])
```

In [ ]:
```
arr3d[0] = old_values

arr3d
```

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

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

In [ ]:
```
x = np.random.randint(0, 10, 2)
y = np.arange(2) + 1
# We can compute inner products between vectors:
print(x)
print(np.dot(x,y))
print(np.dot(y,x))
print(np.outer(x, y))
```

```
[9 8]
25
25
[[ 9 18]
 [ 8 16]]
```

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

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

In [ ]:
```
# Create a sequence of 10 values in range 0 to 5
g = np.linspace(0, 5, 10)
print ("\nA sequential array with 10 values between"
                                        "0 and 5:\n", g)

# Reshaping 3X4 array to 2X2X3 array
arr = np.array([[1, 2, 3, 4],
                [5, 2, 4, 2],
                [1, 2, 0, 1]])

newarr = arr.reshape(2, 2, 3)

print ("\nOriginal array:\n", arr)
print ("Reshaped array:\n", newarr)

# Flatten array
arr = np.array([[1, 2, 3], [4, 5, 6]])
flarr = arr.flatten()

print ("\nOriginal array:\n", arr)
print ("Fattened array:\n", flarr)
```

```
A sequential array with 10 values between0 and 5:
 [0.         0.55555556 1.11111111 1.66666667 2.22222222 2.77777778
 3.33333333 3.88888889 4.44444444 5.        ]

Original array:
 [[1 2 3 4]
 [5 2 4 2]
 [1 2 0 1]]
Reshaped array:
 [[[1 2 3]
  [4 5 2]]

 [[4 2 1]
  [2 0 1]]]

Original array:
```

```
 [[1 2 3]
 [4 5 6]]
Fattened array:
 [1 2 3 4 5 6]
```

In [ ]:

```
arr = np.array([[1, 5, 6],
                [4, 7, 2],
                [3, 1, 9]])

# maximum element of array
print ("Largest element is:", arr.max())
print ("Row-wise maximum elements:",
                    arr.max(axis = 1))

# minimum element of array
print ("Column-wise minimum elements:",
                        arr.min(axis = 0))

# sum of array elements
print ("Sum of all array elements:",
                            arr.sum())

# cumulative sum along each row
print ("Cumulative sum along each row:\n",
                        arr.cumsum(axis = 1))
```

```
Largest element is: 9
Row-wise maximum elements: [6 7 9]
Column-wise minimum elements: [1 1 2]
Sum of all array elements: 38
Cumulative sum along each row:
 [[ 1  6 12]
 [ 4 11 13]
 [ 3  4 13]]
```

In [ ]:

```
a = np.array([[1, 4, 2],
                [3, 4, 6],
              [0, -1, 5]])

# sorted array
print ("Array elements in sorted order:\n",
                    np.sort(a, axis = None))

# sort array row-wise
print ("Row-wise sorted array:\n",
                np.sort(a, axis = 1))

# specify sort algorithm
print ("Column wise sort by applying merge-sort:\n",
            np.sort(a, axis = 0, kind = 'mergesort'))
```

```
Array elements in sorted order:
 [-1  0  1  2  3  4  4  5  6]
Row-wise sorted array:
 [[ 1  2  4]
 [ 3  4  6]
 [-1  0  5]]
Column wise sort by applying merge-sort:
 [[ 0 -1  2]
```

```
[ 1  4  5]
[ 3  4  6]]
```

In [ ]:

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

# horizontal splitting
print("Splitting along horizontal axis into 2 parts:\n", np.hsplit(a, 2))

# vertical splitting
print("\nSplitting along vertical axis into 2 parts:\n", np.vsplit(a, 2))
```

```
Splitting along horizontal axis into 2 parts:
 [array([[1, 3, 5],
       [2, 4, 6]]), array([[ 7,  9, 11],
       [ 8, 10, 12]])]

Splitting along vertical axis into 2 parts:
 [array([[ 1,  3,  5,  7,  9, 11]]), array([[ 2,  4,  6,  8, 10, 12]])]
```

In [ ]:

```
# creating a date
today = np.datetime64('2022-01-20')
print("Date is:", today)
print("Year is:", np.datetime64(today, 'Y'))
```

```
Date is: 2022-01-20
Year is: 2022
```

In [ ]:

```
# creating array of dates in a month
dates = np.arange('2022-01', '2022-02', dtype='datetime64[D]')
print("\nDates of Jan, 2022:\n", dates)
print("Today is Jan:", today in dates)
```

```
Dates of Jan, 2022:
 ['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']
Today is Jan: True
```

In [ ]:

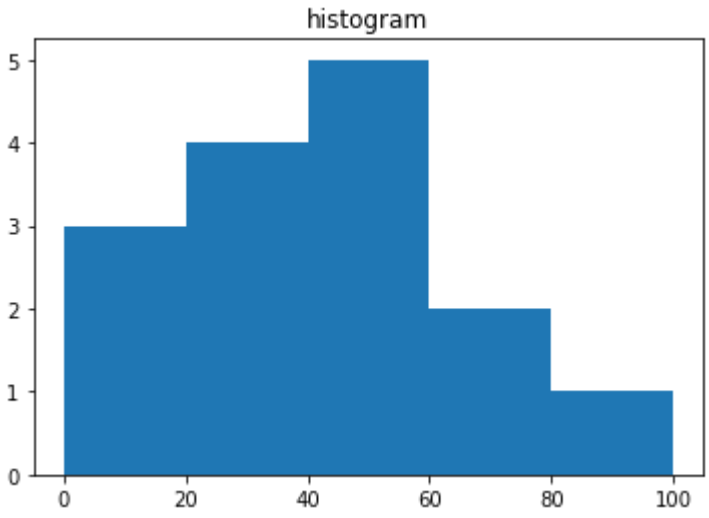
```
A = np.array([[6, 1, 1],
              [4, -2, 5],
              [2, 8, 7]])

print("Rank of A:", np.linalg.matrix_rank(A))

print("\nTrace of A:", np.trace(A))

print("\nDeterminant of A:", np.linalg.det(A))

print("\nInverse of A:\n", np.linalg.inv(A))

print("\nMatrix A raised to power 3:\n", np.linalg.matrix_power(A, 3))
```

```
Rank of A: 3
```

```
Trace of A: 11

Determinant of A: -306.0

Inverse of A:
 [[ 0.17647059 -0.00326797 -0.02287582]
 [ 0.05882353 -0.13071895  0.08496732]
 [-0.11764706  0.1503268   0.05228758]]

Matrix A raised to power 3:
 [[336 162 228]
 [406 162 469]
 [698 702 905]]
```

In [ ]:

```
from matplotlib import pyplot as plt
a = np.array([22,87,5,43,56,73,55,54,11,20,51,5,79,31,27])
plt.hist(a, bins = [0,20,40,60,80,100])
plt.title("histogram")
plt.show()
```



In [ ]:

EDA in Python

> **Steps**
>
> - Understand the Data
> - Clean the Data
> - Fimd a Relationship between data

In [ ]:

```
import plotly.express as px
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
```

In [ ]:
```
df = sns.load_dataset('titanic')
# df = pd.read_csv('/asdf/asdf/titanic.csv')
df.head(5)
```

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

In [ ]:
```
df.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 [ ]:
```
df.shape
```

```
(891, 15)
```

In [ ]:
```
# unique values checking in data
df.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 [ ]:
```
# col names
df.columns
```

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

In [ ]:
```
df['sex'].unique()
```

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

In [ ]:
```
df['age'].unique()
```

```
array([22.  , 38.  , 26.  , 35.  ,   nan, 54.  ,  2.  , 27.  , 14.  ,
        4.  , 58.  , 20.  , 39.  , 55.  , 31.  , 34.  , 15.  , 28.  ,
        8.  , 19.  , 40.  , 66.  , 42.  , 21.  , 18.  ,  3.  ,  7.  ,
       49.  , 29.  , 65.  , 28.5 ,  5.  , 11.  , 45.  , 17.  , 32.  ,
       16.  , 25.  ,  0.83, 30.  , 33.  , 23.  , 24.  , 46.  , 59.  ,
       71.  , 37.  , 47.  , 14.5 , 70.5 , 32.5 , 12.  ,  9.  , 36.5 ,
       51.  , 55.5 , 40.5 , 44.  ,  1.  , 61.  , 56.  , 50.  , 36.  ,
       45.5 , 20.5 , 62.  , 41.  , 52.  , 63.  , 23.5 ,  0.92, 43.  ,
       60.  , 10.  , 64.  , 13.  , 48.  ,  0.75, 53.  , 57.  , 80.  ,
       70.  , 24.5 ,  6.  ,  0.67, 30.5 ,  0.42, 34.5 , 74.  ])
```

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

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

In [ ]:
```
# Assignment
pd.unique(df[['sex', 'who' , 'age', 'fare']].values.ravel('K'))
```

```
array(['male', 'female', 'man', 'woman', 'child', 22.0, 38.0, 26.0, 35.0,
       nan, 54.0, 2.0, 27.0, 14.0, 4.0, 58.0, 20.0, 39.0, 55.0, 31.0,
       34.0, 15.0, 28.0, 8.0, 19.0, 40.0, 66.0, 42.0, 21.0, 18.0, 3.0,
       7.0, 49.0, 29.0, 65.0, 28.5, 5.0, 11.0, 45.0, 17.0, 32.0, 16.0,
       25.0, 0.83, 30.0, 33.0, 23.0, 24.0, 46.0, 59.0, 71.0, 37.0, 47.0,
       14.5, 70.5, 32.5, 12.0, 9.0, 36.5, 51.0, 55.5, 40.5, 44.0, 1.0,
       61.0, 56.0, 50.0, 36.0, 45.5, 20.5, 62.0, 41.0, 52.0, 63.0, 23.5,
       0.92, 43.0, 60.0, 10.0, 64.0, 13.0, 48.0, 0.75, 53.0, 57.0, 80.0,
       70.0, 24.5, 6.0, 0.67, 30.5, 0.42, 34.5, 74.0, 7.25, 71.2833,
       7.925, 53.1, 8.05, 8.4583, 51.8625, 21.075, 11.1333, 30.0708, 16.7,
       26.55, 31.275, 7.8542, 29.125, 7.225, 8.0292, 35.5, 31.3875, 263.0,
       7.8792, 7.8958, 27.7208, 146.5208, 7.75, 10.5, 82.1708, 7.2292,
       11.2417, 9.475, 41.5792, 15.5, 21.6792, 17.8, 39.6875, 7.8,
       76.7292, 61.9792, 27.75, 46.9, 83.475, 27.9, 15.2458, 8.1583,
       8.6625, 73.5, 14.4542, 56.4958, 7.65, 12.475, 9.5, 7.7875, 47.1,
       15.85, 34.375, 61.175, 20.575, 34.6542, 63.3583, 77.2875, 8.6542,
       7.775, 24.15, 9.825, 14.4583, 247.5208, 7.1417, 22.3583, 6.975,
       7.05, 15.0458, 26.2833, 9.2167, 79.2, 6.75, 11.5, 36.75, 7.7958,
       12.525, 66.6, 7.3125, 61.3792, 7.7333, 69.55, 16.1, 15.75, 20.525,
       25.925, 33.5, 30.6958, 25.4667, 28.7125, 0.0, 15.05, 22.025,
       8.4042, 6.4958, 10.4625, 18.7875, 113.275, 76.2917, 90.0, 9.35,
       13.5, 7.55, 26.25, 12.275, 7.125, 52.5542, 20.2125, 86.5, 512.3292,
       79.65, 153.4625, 135.6333, 19.5, 29.7, 77.9583, 20.25, 78.85,
       91.0792, 12.875, 8.85, 151.55, 23.25, 12.35, 110.8833, 108.9,
       56.9292, 83.1583, 262.375, 164.8667, 134.5, 6.2375, 57.9792,
```

```
133.65, 15.9, 9.225, 75.25, 69.3, 55.4417, 211.5, 4.0125, 227.525,
15.7417, 7.7292, 120.0, 12.65, 18.75, 6.8583, 7.875, 14.4, 55.9,
8.1125, 81.8583, 19.2583, 19.9667, 89.1042, 38.5, 7.725, 13.7917,
9.8375, 7.0458, 7.5208, 12.2875, 9.5875, 49.5042, 78.2667, 15.1,
7.6292, 22.525, 26.2875, 59.4, 7.4958, 34.0208, 93.5, 221.7792,
106.425, 49.5, 13.8625, 7.8292, 39.6, 17.4, 51.4792, 26.3875,
40.125, 8.7125, 42.4, 15.55, 32.3208, 7.0542, 8.4333, 25.5875,
9.8417, 8.1375, 10.1708, 211.3375, 13.4167, 7.7417, 9.4833, 7.7375,
8.3625, 23.45, 25.9292, 8.6833, 8.5167, 7.8875, 37.0042, 6.45,
6.95, 8.3, 6.4375, 39.4, 14.1083, 13.8583, 50.4958, 9.8458,
10.5167], dtype=object)
```

Cleaning and Filtering the Data

> Finding missing value Findnig

In [ ]:
```
df.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 [ ]:
```
# droping the col
dff = df.drop(['deck'], axis= 1)
dff.head()
```

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

In [ ]:
```
dff = dff.dropna()
dff.head(2)
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Southar
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	Cherl

In [ ]:
```
dff.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 [ ]:
```
dff['sex'].value_counts()
```

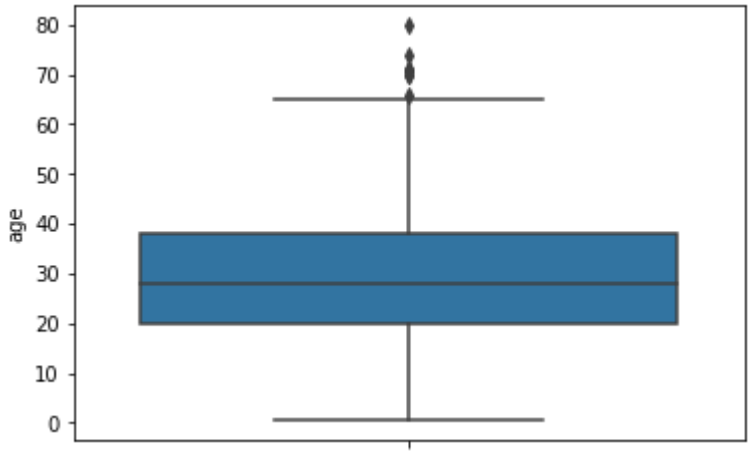
```
male      453
female    259
Name: sex, dtype: int64
```

In [ ]:
```
dff.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

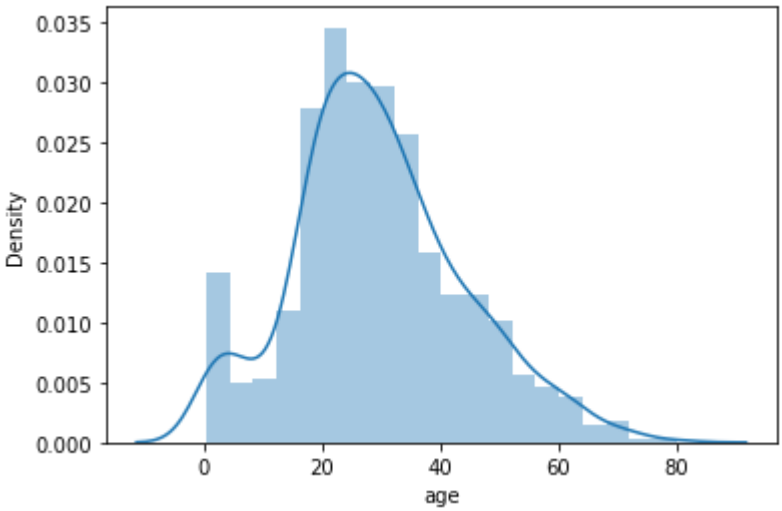
In [ ]:
```
# out lier finding
sns.boxplot( y = 'age', data = dff)#x = 'sex',
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bd2890ac8>
```



In [ ]:
```
sns.distplot(df['age'])# normality check or disperstion zaida hy so for ferfactly data
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

<matplotlib.axes._subplots.AxesSubplot at 0x20bd3006400>
```



In [ ]:
```
dff['age'].mean()
```

```
29.64209269662921
```

In [ ]:
```
dff =  dff[dff['age']< 68]
dff.head()
```

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

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Southar

In [ ]:
```
print(dff.shape)
dff.head(2)
```

```
(705, 14)
```

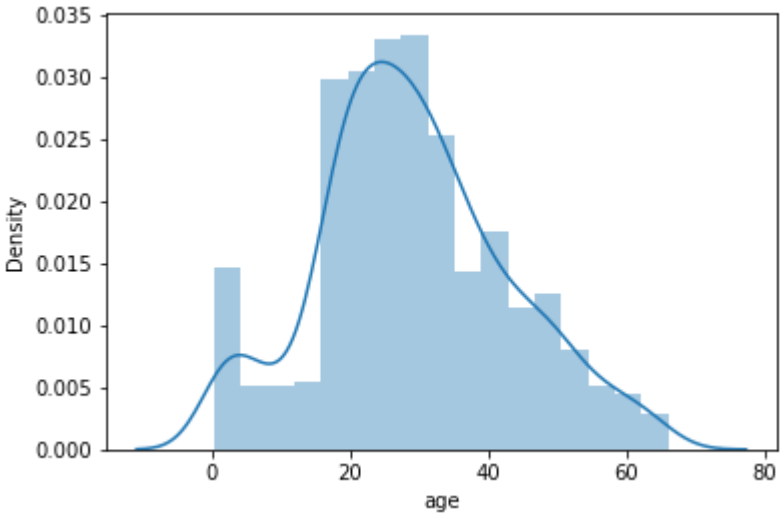
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Southar
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	Cherl

In [ ]:
```
dff.age.value_counts()
```

```
24.00    30
22.00    27
18.00    26
19.00    25
28.00    25
         ..
55.50     1
36.50     1
12.00     1
14.50     1
0.42      1
Name: age, Length: 83, dtype: int64
```

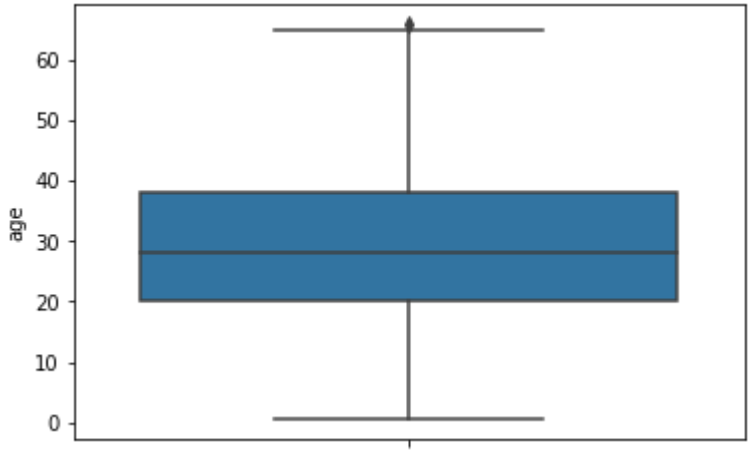
In [ ]:
```
sns.distplot( dff['age'])
```

```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

<matplotlib.axes._subplots.AxesSubplot at 0x20bd30c6278>
```



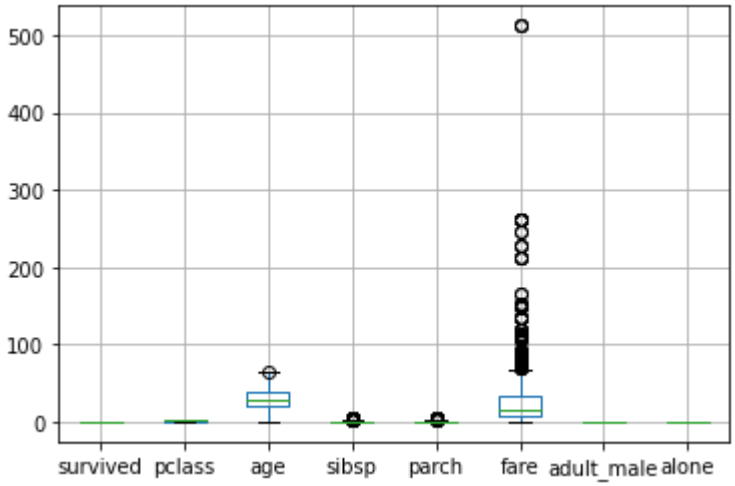
In [ ]:
```
sns.boxplot(y= 'age', data= dff)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bd31694e0>
```



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

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bd31f5e48>
```

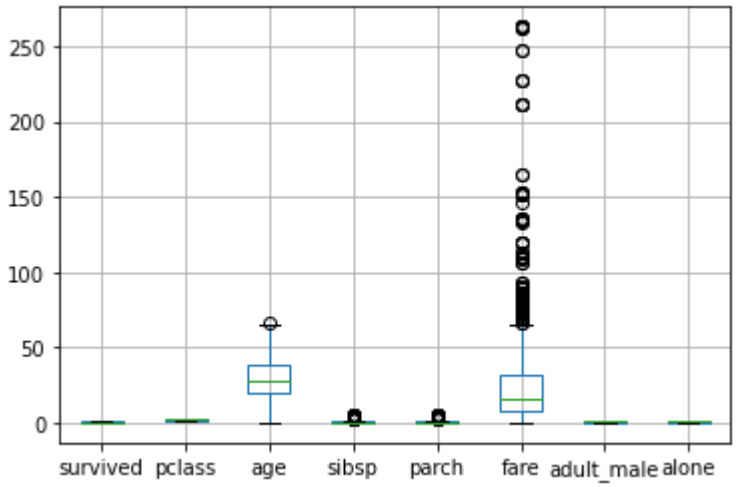


In [ ]:
```
dff = dff[dff['fare']< 300]
dff.head()
```

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

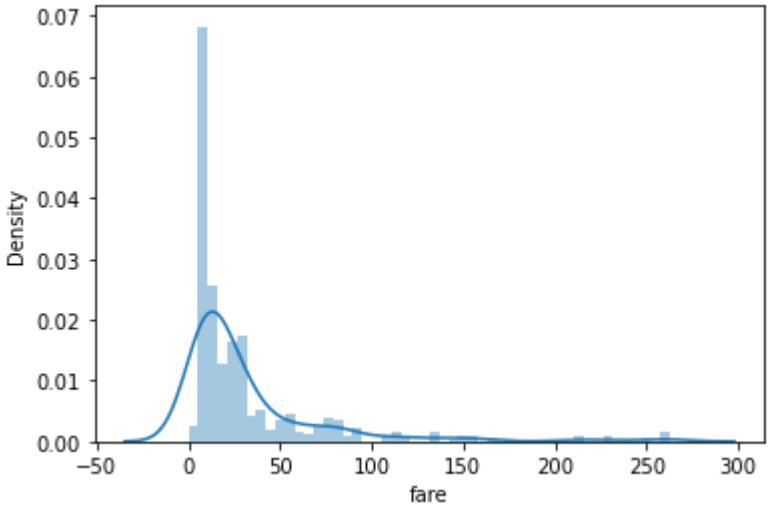
In [ ]:
```
dff.boxplot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bd32de128>
```



In [ ]:
```
sns.distplot(dff['fare'])
```

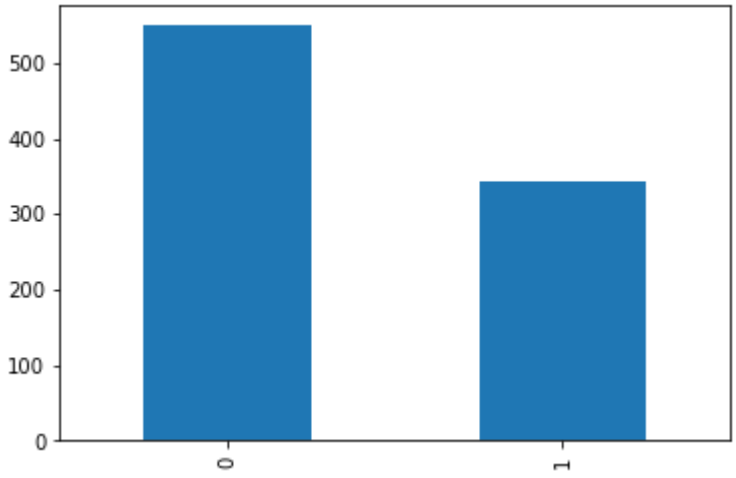
```
C:\Users\Ali\anaconda3\envs\python-chilla\lib\site-packages\seaborn\distributions.py:2619: FutureWarning:

`distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

<matplotlib.axes._subplots.AxesSubplot at 0x20bd342c240>
```



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

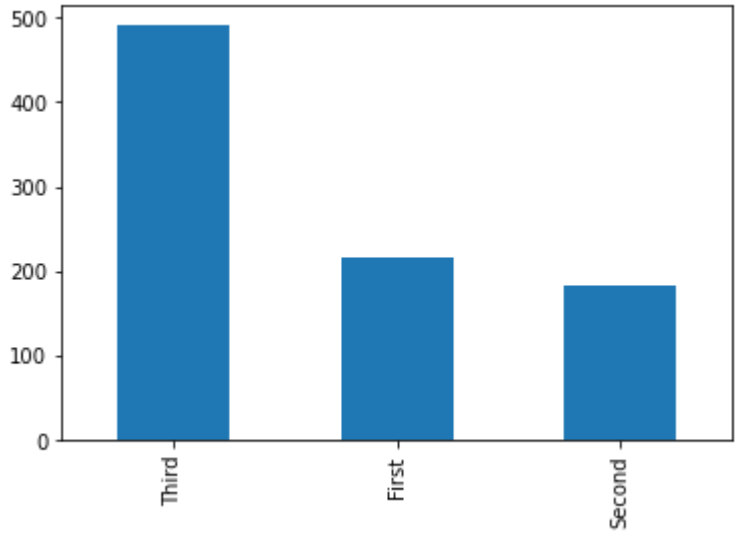
In [ ]:
```
pd.value_counts(df['survived']).plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bd59c66a0>
```



In [ ]:
```
pd.value_counts(df['class']).plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bd5a3a588>
```



In [ ]:
```
dff.groupby(['sex']).mean()
```

sex	survived	pclass	age	sibsp	parch	fare	adult_male	alone
female	0.751938	2.077519	27.717054	0.647287	0.717054	45.530120	0.00000	0.375969
male	0.202703	2.351351	30.048806	0.445946	0.272523	25.038155	0.90991	0.668919

In [ ]:
```
dff.groupby(['sex', 'class']).mean()
```

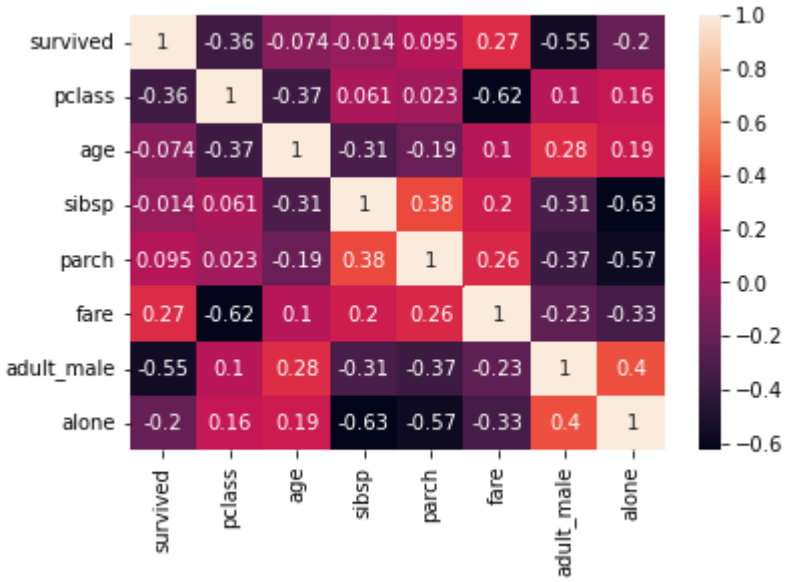
sex	class	survived	pclass	age	sibsp	parch	fare	adult_male	alone
female	**First**	0.963415	1.0	34.231707	0.560976	0.512195	103.696393	0.000000	0.353659
	Second	0.918919	2.0	28.722973	0.500000	0.621622	21.951070	0.000000	0.405405
	Third	0.460784	3.0	21.750000	0.823529	0.950980	15.875369	0.000000	0.372549
male	**First**	0.389474	1.0	40.067579	0.389474	0.336842	62.901096	0.968421	0.526316
	Second	0.153061	2.0	30.340102	0.377551	0.244898	21.221429	0.908163	0.632653
	Third	0.151394	3.0	26.143108	0.494024	0.258964	12.197757	0.888446	0.737052

Relationship or Correlation

In [ ]:
```
cor = dff.corr() #do variable ka relation k interactioon ak k bharny say dosra bhar rah
```

In [ ]:
```
sns.heatmap(cor, annot = True)#only numerical data corelation can be find
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x20bd5d1e7b8>
```



In [ ]:
```
sns.relplot(x = 'age', y= 'sex', hue= 'sex', date = dff)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-51-0267b8f67305> in <module>
----> 1 sns.relplot(x = 'age', y= 'sex', hue= 'sex', date = dff)


~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\_decorators.py in inner_f(*args, **kwargs)
     44             )
     45         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 46         return f(**kwargs)
     47     return inner_f
     48


~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\relational.py in relplot(x, y, hue, size, style, data, row, col, col_wrap, row_order, col_order, palette, hue_order, hue_norm, sizes, size_order, size_norm, markers, dashes, style_order, legend, kind, height, aspect, facet_kws, units, **kwargs)
    948         data=data,
    949         variables=plotter.get_semantics(locals()),
--> 950         legend=legend,
    951     )
    952     p.map_hue(palette=palette, order=hue_order, norm=hue_norm)
```

```
~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\relational.py in __init__(self, data, variables, x_bins, y_bins, estimator, ci, n_boot, alpha, x_jitter, y_jitter, legend)

    585         )
    586
--> 587         super().__init__(data=data, variables=variables)
    588
    589         self.alpha = alpha


~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\_core.py in __init__(self, data, variables)

    603     def __init__(self, data=None, variables={}):
    604
--> 605         self.assign_variables(data, variables)
    606
    607         for var, cls in self._semantic_mappings.items():


~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\_core.py in assign_variables(self, data, variables)

    667             self.input_format = "long"
    668             plot_data, variables = self._assign_variables_longform(
--> 669                 data, **variables,
    670             )
    671


~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\_core.py in _assign_variables_longform(self, data, **kwargs)

    901
    902                 err = f"Could not interpret value `{val}` for parameter `{key}`"
--> 903                 raise ValueError(err)
    904
    905             else:


ValueError: Could not interpret value `sex` for parameter `hue`
```

In [ ]:
```
dff
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	eml
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Sou
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Sou
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	Sou
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Sou
...	...	...	...	...	...	...	...	...	...	...	...	
885	0	3	female	39.0	0	5	29.1250	Q	Third	woman	False	Q
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	Sou
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	Sou
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	Q

702 rows × 14 columns

In [ ]:
```
sns.catplot(x = 'sex', y= 'fare', hue= 'sex', date = dff, kind = 'box')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-59-919ca77c5291> in <module>
----> 1 sns.catplot(x = 'sex', y= 'fare', hue= 'sex', date = dff, kind = 'box')


~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\_decorators.py in inner_f(*args, **kwargs)
     44             )
     45         kwargs.update({k: arg for k, arg in zip(sig.parameters, args)})
---> 46         return f(**kwargs)
     47     return inner_f
     48


~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\categorical.py in catplot(x, y, hue, data, row, col, col_wrap, estimator, ci, n_boot, units, seed, order, hue_order, row_order, col_order, kind, height, aspect, orient, color, palette, legend, legend_out, sharex, sharey, margin_titles, facet_kws, **kwargs)
```

```
   3790     p = _CategoricalPlotter()
   3791     p.require_numeric = plotter_class.require_numeric
-> 3792     p.establish_variables(x_, y_, hue, data, orient, order, hue_order)
   3793     if (
   3794         order is not None
```

~\anaconda3\envs\python-chilla\lib\site-packages\seaborn\categorical.py in establish_variables(self, x, y, hue, data, orient, order, hue_order, units)

```
    151                 if isinstance(var, str):
    152                     err = "Could not interpret input '{}'".format(var)
--> 153                     raise ValueError(err)
    154
    155             # Figure out the plotting orientation
```

```
ValueError: Could not interpret input 'sex'
```

In [ ]: