

01_DS-101_DataScienceCrashCourse

July 23, 2022

0.1 Basics Operators

```
[ ]: print(1+2)
      print(3-2)
      print(3*2)
      print(4/2)
      print(9%2)
      print(7//2)
```

```
3
1
6
2.0
1
3
```

0.2 Strings

```
[ ]: print('Single Quote Test')
      print('Double Quotes Test')
      print('Triple Quotes Test')

      print("What's up")
```

```
Single Quote Test
Double Quotes Test
Triple Quotes Test
What's up
```

0.3 Variables

Rules to assign a variable: 1. The variable should contain letters, numbers or underscores 2. Do not start with numbers 3. Spaces are not allowed 4. Do not use keywords used in functions (break, mean, test etc.) 5. Short and descriptive 6. Case sensitive

```
[ ]: # Variables: Objects containing Specific Values
      x=5
      print(x)
```

```
x=15
print(x)

print(type(x))
```

```
5
15
<class 'int'>
```

0.4 Input Variables

```
[ ]: # input function is used to get user input

fruit_basket = input('What is your favorite fruit?')
fruit_basket
```

```
[ ]: 'Mango'
```

```
[ ]: name = input('What is your Name?')
greeting = 'Hello!'
print(greeting, name)
```

Hello! Muhammad Waleed Anjum

0.5 Conditional Logics

- equal to ==
- not equal to !=
- less than <
- greater than >
- less than and equal to <=
- greater than and equal to >=

```
[ ]: age_at_school = 5
ahmed_age = int(input('What is Ahmed Age? '))
print(age_at_school==ahmed_age)
```

True

0.6 Type Conversion

```
[ ]: x = 10          # integer
y = 10.2          # Float
z = 'Hello'       # String

# Implicit type conversion
x = x + y
print(x, 'Type of x is: ', type(x))
```

```
# Explicit Type Conversion
age = int(input('What is your Age? '))
print(age, type(age))
```

20.2 Type of x is: <class 'float'>
 32 <class 'int'>

0.7 if else elif statements

```
[ ]: req_age_at_school = 5
req_age_at_college = 14
ahmed_age = int(input('What is Ahmed Age? '))
# Can Ahmed go to school?
if ahmed_age >= req_age_at_college:
    print(f"Ahmed Age is: {ahmed_age}\nYes, Ahmed Can go to college")
elif ahmed_age >= req_age_at_school:
    print(f"Ahmed Age is: {ahmed_age}\nYes, Ahmed Can go to School")
else:
    print(f"Ahmed Age is: {ahmed_age}\nSorry, Ahmed is underage")
```

Ahmed Age is: 16
 Yes, Ahmed Can go to college

0.8 Functions

```
[ ]: # Defining a function (def)
# Method 1
def print_code():
    print('We are learning with babaAmmar')
    print('We are learning with babaAmmar')
    print('We are learning with babaAmmar')

# Method 1
def print_code2():
    text = 'We are learning with babaAmmar'
    print(text)
    print(text)
    print(text)

print_code()
print('-----')
print_code2()
```

We are learning with babaAmmar
 We are learning with babaAmmar
 We are learning with babaAmmar

 We are learning with babaAmmar

We are learning with babaAmmar
We are learning with babaAmmar

```
[ ]: # School Age Calculator
def school_age_calc(age):
    if age == 5:
        print(f"Ahmed Age is: {age}\nAhmed Can join school")
    elif age > 5:
        print(f'Ahmed Age is: {age}\nAhmed should go to Higher School')
    else:
        print(f"Ahmed Age is: {age}\nSorry, Ahmed is underage")

school_age_calc(3)
```

Ahmed Age is: 3
Sorry, Ahmed is underage

```
[ ]: # Defining a function of future
def future_age(age):
    new_age = age + 20
    return new_age

print('Future age is: ',future_age(18))
```

Future age is: 38

0.9 Loops

```
[ ]: # While Loop

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

0
1
2
3
4
5

```
[ ]: # For Loop

for x in range(5,10):
    print(x)
```

5
6
7

8
9

```
[ ]: # Array
days= ['Mon', 'Tues', 'Wed', 'Thur', 'Fri', 'Sat', 'Sun']

for d in days:
    #if (d == 'Fri'): break      #loop stops
    #if (d == 'Fri'): continue  #skips d
    print(d)
```

Mon
Tues
Wed
Thur
Fri
Sat
Sun

0.10 Import Libraries

```
[ ]: # library is already developed codes
import math
print('The Value of pi is: ', math.pi)

import statistics
x = [150, 250, 350, 450]
print('Mean Value of x is: ',statistics.mean(x))
```

The Value of pi is: 3.141592653589793
Mean Value of x is: 300

0.11 Trouble Shooting

```
[ ]: print(25/0)      # runtime error
print(we are learning python)      # syntax error
```

0.12 Indexing

```
[ ]: a = 'samosa pakora'
print('First index is: ',a[0])      # Index starts from 0

# Length of indices
print('Length(Total Characters) of a are: ',len(a))
```

First index is: s
Length(Total Characters) of a are: 13

```
[ ]: print('a is: ',a)
      print('a[0:5]',a[0:5])
      print('a[0:13]',a[0:13])    # last index is exclusive
      print('a[:5]',a[:5])
      print('a[-5]',a[-5])
      print('a[0:-5]',a[0:-5])
      print('a[-6:-1]',a[-6:-1])
```

```
a is:  samosa pakora
a[0:5] samos
a[0:13] samosa pakora
a[:5] samos
a[-5] a
a[0:-5] samosa p
a[-6:-1] pakor
```

0.13 String Methods

```
[ ]: food = 'biryani'

      print('Capitalize: ', food.capitalize())
      print('UpperCase: ', food.upper())
      print('LowerCase: ', food.lower())

      # Replace
      print('Replacing b with Sh: ', food.replace('b', 'Sh'))

      # Counting Specific alphabat in a string
      name = 'Muhammad Waleed Anjum'
      print(f"Number of 'a' in {name} are: ", name.count('a'))

      # Finding index number in String
      name = 'Muhammad Waleed Anjum'
      print("Index number of 'h' is: ",name.find('h'))
      print("Index number of 'a' is: ",name.find('a'))
      print("Index number of 'ee' is: ",name.find('ee'))

      # How to split a string
      food = 'I love, samosa, pakora, biryani'
      print('Splitting string based on specific charcter', food.split(','))
```

```
Capitalize:  Biryani
UpperCase:  BIRYANI
LowerCase:  biryani
Replacing b with Sh:  Shiryani
Number of 'a' in Muhammad Waleed Anjum are:  3
Index number of 'h' is:  2
Index number of 'a' is:  3
```

Index number of 'ee' is: 12

Splitting string based on specific charcter ['I love', ' samosa', ' pakora', ' biryani']

0.14 Basic Data Structure in Python

There are four basic Data Structures 1. Tuple 2. List 3. Dictionaries 4. Set

0.14.1 1. Tuple

- Ordered Collection of elements
- Enclosed in small brackets()
- Different kind of data can be stored
- Unmutatble

```
[ ]: tup = (1, 'python', 3.5, True)
print(type(tup))
print('Number of elements in Tuple: ',len(tup))

tup2 = (2, 'Tuple', False)

concat = tup + tup2
print('Concatenation of two tuples: ', concat)

tup3 = (10, 723, 43, 11, 53)
print('Maximum Number in tup3 is: ', max(tup3))
```

<class 'tuple'>

Number of elements in Tuple: 4

Concatenation of two tuples: (1, 'python', 3.5, True, 2, 'Tuple', False)

Maximum Number in tup3 is: 723

0.14.2 2. List

- Ordered Collection of Elements
- enclosed in [] brackets
- mutable

```
[ ]: list1 = [2, 'waleed', True]
print('list1 is: ', list1)

print(type(list1))

list1.reverse()
print('Reverse elements', list1)

list1.append('Pakistan')
print('Append something in list: ', list1)
```

```
print('Counting Something in list: ',list1.count(3))
```

```
list2 = [1,87,34,23,96,34]
```

```
print('list2 is: ', list2)
```

```
list2.sort()
```

```
print('Sorted list: ', list2)
```

```
list1 is: [2, 'waleed', True]
```

```
<class 'list'>
```

```
Reverse elements [True, 'waleed', 2]
```

```
Append something in list: [True, 'waleed', 2, 'Pakistan']
```

```
Counting Something in list: 0
```

```
list2 is: [1, 87, 34, 23, 96, 34]
```

```
Sorted list: [1, 23, 34, 34, 87, 96]
```

0.14.3 3. Dictionaries

- UnOrdered Collection of elements
- key and value
- Enclosed in curly brackets {}
- Unmutatble

```
[ ]: d1 = {'samosa': 30,
'pakora': 50,
'Raita': 40,
'Roll': 100}

print('Dictionary d1 is: ',d1)
type(d1)

# extract data
k = d1.keys()
v = d1.values()
print('Keys in d1 are: ',k)
print('Values in d1 are: ',v)

# Adding new element
d1['shawarma'] = 120
print('After adding new element in d1 Dictionary is: ',d1)

# Updating existing value
d1['shawarma'] = 160
print('Updated d1 Dictionary is: ',d1)

# Concatenating two dictionaries
d2 = {'biryani': 200, 'Pulao': 160}

d1.update(d2)
```



```
print('concatenated dictionary: ', d1)
```

Dictionary d1 is: {'samosa': 30, 'pakora': 50, 'Raita': 40, 'Roll': 100}
Keys in d1 are: dict_keys(['samosa', 'pakora', 'Raita', 'Roll'])
Values in d1 are: dict_values([30, 50, 40, 100])
After adding new element in d1 Dictionary is: {'samosa': 30, 'pakora': 50, 'Raita': 40, 'Roll': 100, 'shawarma': 120}
Updated d1 Dictionary is: {'samosa': 30, 'pakora': 50, 'Raita': 40, 'Roll': 100, 'shawarma': 160}
concatenated dictionary: {'samosa': 30, 'pakora': 50, 'Raita': 40, 'Roll': 100, 'shawarma': 160, 'biryani': 200, 'Pulao': 160}

0.14.4 4. Set

- UnOrdered and unindexed
- Enclosed in only curly brackets {}
- No duplicates allowed

```
[ ]: s1 = {1, 2.5, 7, 'waleed', 'Pakistan', True} # Sets cannot read boolean
      ↪ operators
print('Set s1 is: ', s1)

# Adding same value again
s1.add('waleed')
print('Duplicates not allowed: ', s1)
```

Set s1 is: {1, 2.5, 'waleed', 7, 'Pakistan'}
Duplicates not allowed: {1, 2.5, 'waleed', 7, 'Pakistan'}

0.15 Numpy (Numerical Python)

```
[ ]: # pip install numpy (installation)
      # importing numpy
      import numpy as np

      # Creating an array using numpy
      food = np.array(['Pakora', 'Samosa', 'Raita'])
      print('food array: ', food)

      price = np.array([5, 5, 5])
      print('Data type of array: ', type(price))
      print('length of food array: ', len(food))
```

food array: ['Pakora' 'Samosa' 'Raita']
Data type of array: <class 'numpy.ndarray'>
length of food array: 3

```
[ ]: # zeros method
      print('Zeros: ', np.zeros(5))
```

```

# ones
print('Ones: ', np.ones(5))

# empty
print('empty: ', np.empty(5))

# arange
print('arange: ', np.arange(6))
print('arange with specific start and end: ', np.arange(2, 10))
print('arange with specific interval: ', np.arange(2, 20, 3))

# linspace
print('linspace (same interval): ', np.linspace(1, 100, num=10))

```

```

Zeros: [0. 0. 0. 0. 0.]
Ones: [1. 1. 1. 1. 1.]
empty: [1. 1. 1. 1. 1.]
arange: [0 1 2 3 4 5]
arange with specific start and end: [2 3 4 5 6 7 8 9]
arange with specific interval: [ 2  5  8 11 14 17]
linspace (same interval): [ 1. 12. 23. 34. 45. 56. 67. 78. 89. 100.]

```

```

[ ]: # specify data type
print('Array in int: ', np.ones(10, dtype=int))
print('Array in float: ', np.ones(10, dtype=float))

```

```

Array in int: [1 1 1 1 1 1 1 1 1 1]
Array in float: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]

```

```

[ ]: # Array functions
# 1-D Array
a = np.array([10, 12, 15, 65, 34, 93, 10.4])
print('array a: ', a)
a.sort()
print('Sorted a Array: ', a)

b = np.array([13, 52, 15, 69, 34, 91, 10.4])
concat = np.concatenate((a,b))
print('Concatenated Arrays', concat)

```

```

array a: [10. 12. 15. 65. 34. 93. 10.4]
Sorted a Array: [10. 10.4 12. 15. 34. 65. 93. ]
Concatenated Arrays [10. 10.4 12. 15. 34. 65. 93. 13. 52. 15. 69. 34.
91. 10.4]

```

```

[ ]: # 2-D Array
a = np.array([[0,1,2,3,4], [5,6,7,8,9]])
b = np.array([[0,1,2,3,4], [5,6,7,8,9]])

```

```

print('a Array: \n', a)
print('\nb Array: \n', b)

c = np.concatenate((a,b), axis=0) # Along Rows
d = np.concatenate((a,b), axis=1) # Along Columns
print('\nc Array: \n', c)
print('\nd Array: \n', d)

```

```

a Array:
[[0 1 2 3 4]
 [5 6 7 8 9]]

```

```

b Array:
[[0 1 2 3 4]
 [5 6 7 8 9]]

```

```

c Array:
[[0 1 2 3 4]
 [5 6 7 8 9]
 [0 1 2 3 4]
 [5 6 7 8 9]]

```

```

d Array:
[[0 1 2 3 4 0 1 2 3 4]
 [5 6 7 8 9 5 6 7 8 9]]

```

```

[ ]: # 3-D Array
a = np.array([[0,1,2,3],
              [4,5,6,7]],
              [[0,1,2,3],
               [4,5,6,7]],
              [[0,1,2,3],
               [4,5,6,7]])
print('3D a Array: \n',a)

print('\nDimension of An Array: ', a.ndim)
print('Size of An Array: ', a.size) # Number of elements
print('Shape of An Array: ', a.shape) # (dimension, rows, columns)

```

```

3D a Array:
[[[0 1 2 3]
  [4 5 6 7]]

```

```

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

```

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

Dimension of An Array: 3

Size of An Array: 24

Shape of An Array: (3, 2, 4)

```
[ ]: # Reshape Method
# 2-D Array by using Reshape
a = np.arange(9)
b = a.reshape(3,3)
print('2-D Array by using Reshape:\n', b)

# 3-D Array by using Reshape
x = np.arange(24)
y = x.reshape(3,2,4)
print('\n3-D Array by using Reshape:\n', y)
```

2-D Array by using Reshape:

```
[[0 1 2]
 [3 4 5]
 [6 7 8]]
```

3-D Array by using Reshape:

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

```
[[ 8  9 10 11]
 [12 13 14 15]]
```

```
[[16 17 18 19]
 [20 21 22 23]]]
```

```
[ ]: # Converting Array dimensions
# 1-D to 2-D
a = np.array([1,2,3,4,5,6,7,8,9])
print('Array a: ',a)
print('Dimension of a Array: ', a.ndim)
print('Shape of An a Array: ', a.shape)

b = a[np.newaxis, :] # Row Conversion
print('\nArray b (Converted to 2D): ',b)
print('Dimensions of b Array: ', b.ndim)
print('Shape of b Array: ', b.shape)

c = a[:, np.newaxis] # Column Conversion
print('\nArray c (Converted to 2D): ',c)
print('Dimensions of c Array: ', c.ndim)
```

```
print('Shape of c Array: ', c.shape)
```

Array a: [1 2 3 4 5 6 7 8 9]

Dimension of a Array: 1

Shape of An a Array: (9,)

Array b (Converted to 2D): [[1 2 3 4 5 6 7 8 9]]

Dimensions of b Array: 2

Shape of b Array: (1, 9)

Array c (Converted to 2D): [[1]

[2]

[3]

[4]

[5]

[6]

[7]

[8]

[9]]

Dimensions of c Array: 2

Shape of c Array: (9, 1)

```
[ ]: # Different operations
a = np.array([1,2,3,4,5,6,7,8,9])
print('a*6: ', a*6)
print('sum: ', a.sum())
print('Mean: ', a.mean())
```

a*6: [6 12 18 24 30 36 42 48 54]

sum: 45

Mean: 5.0

0.16 Pandas (Pannel Data Analysis)

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

```
[ ]: # Import libraries
import pandas as pd
import numpy as np
```

```
[ ]: # object creation
s = pd.Series([1,3, np.nan,5,7,8,9])
print('Series s:\n',s)
```

Series s:

0 1.0

1 3.0

2 NaN

3 5.0

```
4    7.0
5    8.0
6    9.0
dtype: float64
```

```
[ ]: dates = pd.date_range('20220101', periods=20)
      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'],
                    dtype='datetime64[ns]', freq='D')
```

```
[ ]: # DataFrame
      df = pd.DataFrame(np.random.randn(20, 4), index=dates, columns=list('ABCD'))
      df
```

```
[ ]:
      A          B          C          D
2022-01-01 -0.950106  0.666944  1.255174  0.620168
2022-01-02  0.679854  0.704714  1.847270  0.444161
2022-01-03 -0.549860  0.095673 -1.062794 -1.483643
2022-01-04 -0.840756 -1.541808  0.716966 -0.547674
2022-01-05  1.903297  0.829506 -0.492042  1.245799
2022-01-06 -2.087375 -1.134863  1.065108 -0.817172
2022-01-07 -0.333677 -0.782179  1.137580 -1.441844
2022-01-08 -0.588446  0.294438  1.343623  0.301845
2022-01-09 -0.096240 -1.650489  0.090041  0.954330
2022-01-10 -0.649596  0.941632  0.801956  0.756375
2022-01-11  0.593017  1.434123  1.115778 -0.751717
2022-01-12 -0.231228  1.241760 -2.217075 -1.623309
2022-01-13  0.810533  2.313326 -0.747402 -1.227251
2022-01-14 -0.446317  0.356303  0.430054  1.050609
2022-01-15 -1.143818 -0.885201  2.788002  0.056081
2022-01-16 -0.129958 -0.095443  1.490171 -1.012446
2022-01-17 -0.202072  0.562094  0.817439 -1.958470
2022-01-18 -0.002977  2.166893 -0.416769 -0.883259
2022-01-19 -1.469375 -0.034409 -0.486798 -1.216058
2022-01-20  0.297953  0.560134 -0.049372  0.240628
```

```
[ ]: df2 = pd.DataFrame(
      {
          'A': 1.0,
          'B' : pd.Timestamp('20220314'),
          'C' : pd.Series(1, index=list(range(4)), dtype='float32'),
          'D' : np.array([3] * 4, dtype='int32'),
          'E' : pd.Categorical(['girl', 'woman', 'girl', 'woman']),
      })
```

```

        'F' : 'female',
    }
)
df2

```

```

[ ]:
   A      B      C  D      E      F
0  1.0 2022-03-14  1.0  3  girl  female
1  1.0 2022-03-14  1.0  3  woman female
2  1.0 2022-03-14  1.0  3  girl  female
3  1.0 2022-03-14  1.0  3  woman female

```

```

[ ]: df2.dtypes

```

```

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

```

```

[ ]: # head method
     df.head(4) # shows first 4 rows

```

```

[ ]:
      A      B      C      D
2022-01-01 -0.950106  0.666944  1.255174  0.620168
2022-01-02  0.679854  0.704714  1.847270  0.444161
2022-01-03 -0.549860  0.095673 -1.062794 -1.483643
2022-01-04 -0.840756 -1.541808  0.716966 -0.547674

```

```

[ ]: df.tail(2) # last 2 rows

```

```

[ ]:
      A      B      C      D
2022-01-19 -1.469375 -0.034409 -0.486798 -1.216058
2022-01-20  0.297953  0.560134 -0.049372  0.240628

```

```

[ ]: df.index

```

```

[ ]: 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')

```

```

[ ]: # convert into numpy array
     df.to_numpy()

```

```
[ ]: array([[ -0.95010582,  0.66694432,  1.25517431,  0.62016755],
 [ 0.67985399,  0.70471366,  1.84726953,  0.44416054],
 [-0.54985992,  0.0956732 , -1.06279356, -1.48364278],
 [-0.84075575, -1.54180846,  0.71696612, -0.54767372],
 [ 1.90329718,  0.82950604, -0.49204199,  1.24579925],
 [-2.0873748 , -1.13486316,  1.06510831, -0.8171724 ],
 [-0.3336765 , -0.782179 ,  1.13757981, -1.44184412],
 [-0.58844572,  0.29443818,  1.34362263,  0.30184502],
 [-0.09623966, -1.65048853,  0.09004135,  0.95432994],
 [-0.64959576,  0.94163212,  0.80195586,  0.75637494],
 [ 0.59301666,  1.43412276,  1.11577752, -0.75171684],
 [-0.2312285 ,  1.24176027, -2.21707466, -1.62330856],
 [ 0.8105335 ,  2.31332632, -0.7474015 , -1.22725142],
 [-0.44631672,  0.35630299,  0.43005431,  1.05060875],
 [-1.14381787, -0.88520068,  2.78800203,  0.05608148],
 [-0.12995805, -0.09544285,  1.49017146, -1.0124465 ],
 [-0.2020721 ,  0.56209436,  0.81743896, -1.95846952],
 [-0.00297664,  2.16689279, -0.41676885, -0.88325934],
 [-1.46937519, -0.03440855, -0.48679757, -1.21605849],
 [ 0.29795254,  0.56013394, -0.04937241,  0.2406284 ]])
```

```
[ ]: df2.to_numpy()
```

```
[ ]: array([[1.0, Timestamp('2022-03-14 00:00:00'), 1.0, 3, 'girl', 'female'],
 [1.0, Timestamp('2022-03-14 00:00:00'), 1.0, 3, 'woman', 'female'],
 [1.0, Timestamp('2022-03-14 00:00:00'), 1.0, 3, 'girl', 'female'],
 [1.0, Timestamp('2022-03-14 00:00:00'), 1.0, 3, 'woman', 'female']],
      dtype=object)
```

```
[ ]: # describe function
df.describe()
```

```
[ ]:
count    20.000000    20.000000    20.000000    20.000000
mean     -0.271857     0.302157     0.471346    -0.364642
std       0.877787     1.099836     1.153807     1.007529
min      -2.087375    -1.650489    -2.217075    -1.958470
25%      -0.697386    -0.267127    -0.434276    -1.218857
50%      -0.282453     0.458218     0.759461    -0.649695
75%       0.072256     0.857538     1.166978     0.488162
max       1.903297     2.313326     2.788002     1.245799
```

```
[ ]: # transpose data
df2.T
```

```
[ ]:
          0          1          2  \
A          1.0          1.0          1.0
B  2022-03-14 00:00:00  2022-03-14 00:00:00  2022-03-14 00:00:00
```


C	1.0	1.0	1.0
D	3	3	3
E	girl	woman	girl
F	female	female	female

	3
A	1.0
B	2022-03-14 00:00:00
C	1.0
D	3
E	woman
F	female

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

```
[ ]:
```

	A	B	C	D
2022-01-01	-0.950106	0.666944	1.255174	0.620168
2022-01-02	0.679854	0.704714	1.847270	0.444161
2022-01-03	-0.549860	0.095673	-1.062794	-1.483643
2022-01-04	-0.840756	-1.541808	0.716966	-0.547674
2022-01-05	1.903297	0.829506	-0.492042	1.245799
2022-01-06	-2.087375	-1.134863	1.065108	-0.817172
2022-01-07	-0.333677	-0.782179	1.137580	-1.441844
2022-01-08	-0.588446	0.294438	1.343623	0.301845
2022-01-09	-0.096240	-1.650489	0.090041	0.954330
2022-01-10	-0.649596	0.941632	0.801956	0.756375
2022-01-11	0.593017	1.434123	1.115778	-0.751717
2022-01-12	-0.231228	1.241760	-2.217075	-1.623309
2022-01-13	0.810533	2.313326	-0.747402	-1.227251
2022-01-14	-0.446317	0.356303	0.430054	1.050609
2022-01-15	-1.143818	-0.885201	2.788002	0.056081
2022-01-16	-0.129958	-0.095443	1.490171	-1.012446
2022-01-17	-0.202072	0.562094	0.817439	-1.958470
2022-01-18	-0.002977	2.166893	-0.416769	-0.883259
2022-01-19	-1.469375	-0.034409	-0.486798	-1.216058
2022-01-20	0.297953	0.560134	-0.049372	0.240628

```
[ ]: df.sort_index(axis=0, ascending=True)
```

```
[ ]:
```

	A	B	C	D
2022-01-01	-0.950106	0.666944	1.255174	0.620168
2022-01-02	0.679854	0.704714	1.847270	0.444161
2022-01-03	-0.549860	0.095673	-1.062794	-1.483643
2022-01-04	-0.840756	-1.541808	0.716966	-0.547674
2022-01-05	1.903297	0.829506	-0.492042	1.245799
2022-01-06	-2.087375	-1.134863	1.065108	-0.817172
2022-01-07	-0.333677	-0.782179	1.137580	-1.441844

```

2022-01-08 -0.588446  0.294438  1.343623  0.301845
2022-01-09 -0.096240 -1.650489  0.090041  0.954330
2022-01-10 -0.649596  0.941632  0.801956  0.756375
2022-01-11  0.593017  1.434123  1.115778 -0.751717
2022-01-12 -0.231228  1.241760 -2.217075 -1.623309
2022-01-13  0.810533  2.313326 -0.747402 -1.227251
2022-01-14 -0.446317  0.356303  0.430054  1.050609
2022-01-15 -1.143818 -0.885201  2.788002  0.056081
2022-01-16 -0.129958 -0.095443  1.490171 -1.012446
2022-01-17 -0.202072  0.562094  0.817439 -1.958470
2022-01-18 -0.002977  2.166893 -0.416769 -0.883259
2022-01-19 -1.469375 -0.034409 -0.486798 -1.216058
2022-01-20  0.297953  0.560134 -0.049372  0.240628

```

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

```

[ ]:
      A      B      C      D
2022-01-09 -0.096240 -1.650489  0.090041  0.954330
2022-01-04 -0.840756 -1.541808  0.716966 -0.547674
2022-01-06 -2.087375 -1.134863  1.065108 -0.817172
2022-01-15 -1.143818 -0.885201  2.788002  0.056081
2022-01-07 -0.333677 -0.782179  1.137580 -1.441844
2022-01-16 -0.129958 -0.095443  1.490171 -1.012446
2022-01-19 -1.469375 -0.034409 -0.486798 -1.216058
2022-01-03 -0.549860  0.095673 -1.062794 -1.483643
2022-01-08 -0.588446  0.294438  1.343623  0.301845
2022-01-14 -0.446317  0.356303  0.430054  1.050609
2022-01-20  0.297953  0.560134 -0.049372  0.240628
2022-01-17 -0.202072  0.562094  0.817439 -1.958470
2022-01-01 -0.950106  0.666944  1.255174  0.620168
2022-01-02  0.679854  0.704714  1.847270  0.444161
2022-01-05  1.903297  0.829506 -0.492042  1.245799
2022-01-10 -0.649596  0.941632  0.801956  0.756375
2022-01-12 -0.231228  1.241760 -2.217075 -1.623309
2022-01-11  0.593017  1.434123  1.115778 -0.751717
2022-01-18 -0.002977  2.166893 -0.416769 -0.883259
2022-01-13  0.810533  2.313326 -0.747402 -1.227251

```

```
[ ]: df['A']
```

```

[ ]: 2022-01-01    -0.950106
      2022-01-02     0.679854
      2022-01-03    -0.549860
      2022-01-04    -0.840756
      2022-01-05     1.903297
      2022-01-06    -2.087375
      2022-01-07    -0.333677

```

```

2022-01-08    -0.588446
2022-01-09    -0.096240
2022-01-10    -0.649596
2022-01-11     0.593017
2022-01-12    -0.231228
2022-01-13     0.810533
2022-01-14    -0.446317
2022-01-15    -1.143818
2022-01-16    -0.129958
2022-01-17    -0.202072
2022-01-18    -0.002977
2022-01-19    -1.469375
2022-01-20     0.297953
Freq: D, Name: A, dtype: float64

```

```
[ ]: # row wise selection
df[0:2]
```

```
[ ]:
           A         B         C         D
2022-01-01 -0.950106  0.666944  1.255174  0.620168
2022-01-02  0.679854  0.704714  1.847270  0.444161

```

```
[ ]: # select by label
df.loc[dates[0]]
```

```
[ ]: A    -0.950106
      B     0.666944
      C     1.255174
      D     0.620168
      Name: 2022-01-01 00:00:00, dtype: float64

```

```
[ ]: # column wise selection
df.loc[:, ['A', 'B']]
```

```
[ ]:
           A         B
2022-01-01 -0.950106  0.666944
2022-01-02  0.679854  0.704714
2022-01-03 -0.549860  0.095673
2022-01-04 -0.840756 -1.541808
2022-01-05  1.903297  0.829506
2022-01-06 -2.087375 -1.134863
2022-01-07 -0.333677 -0.782179
2022-01-08 -0.588446  0.294438
2022-01-09 -0.096240 -1.650489
2022-01-10 -0.649596  0.941632
2022-01-11  0.593017  1.434123
2022-01-12 -0.231228  1.241760

```

```

2022-01-13  0.810533  2.313326
2022-01-14 -0.446317  0.356303
2022-01-15 -1.143818 -0.885201
2022-01-16 -0.129958 -0.095443
2022-01-17 -0.202072  0.562094
2022-01-18 -0.002977  2.166893
2022-01-19 -1.469375 -0.034409
2022-01-20  0.297953  0.560134

```

```
[ ]: df.loc['20220102':'20220104', ['A', 'B']]
```

```
[ ]:
      A      B
2022-01-02  0.679854  0.704714
2022-01-03 -0.549860  0.095673
2022-01-04 -0.840756 -1.541808

```

```
[ ]: df.loc['20220102', ['A', 'B', 'C']]
```

```
[ ]: A    0.679854
      B    0.704714
      C    1.847270
      Name: 2022-01-02 00:00:00, dtype: float64

```

```
[ ]: #specify value based on date
      df.at[dates[0], 'A']
```

```
[ ]: -0.9501058169930408
```

```
[ ]: # iloc function
      df.iloc[3]
```

```
[ ]: A    -0.840756
      B   -1.541808
      C    0.716966
      D   -0.547674
      Name: 2022-01-04 00:00:00, dtype: float64

```

```
[ ]: #implicit
      df.iloc[0:5, 0:3]
```

```
[ ]:
      A      B      C
2022-01-01 -0.950106  0.666944  1.255174
2022-01-02  0.679854  0.704714  1.847270
2022-01-03 -0.549860  0.095673 -1.062794
2022-01-04 -0.840756 -1.541808  0.716966
2022-01-05  1.903297  0.829506 -0.492042

```

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

```
[ ]:
```

	A	B
2022-01-01	-0.950106	0.666944
2022-01-02	0.679854	0.704714
2022-01-03	-0.549860	0.095673
2022-01-04	-0.840756	-1.541808
2022-01-05	1.903297	0.829506
2022-01-06	-2.087375	-1.134863
2022-01-07	-0.333677	-0.782179
2022-01-08	-0.588446	0.294438
2022-01-09	-0.096240	-1.650489
2022-01-10	-0.649596	0.941632
2022-01-11	0.593017	1.434123
2022-01-12	-0.231228	1.241760
2022-01-13	0.810533	2.313326
2022-01-14	-0.446317	0.356303
2022-01-15	-1.143818	-0.885201
2022-01-16	-0.129958	-0.095443
2022-01-17	-0.202072	0.562094
2022-01-18	-0.002977	2.166893
2022-01-19	-1.469375	-0.034409
2022-01-20	0.297953	0.560134

```
[ ]: df[df['A'] > 0]
```

```
[ ]:
```

	A	B	C	D
2022-01-02	0.679854	0.704714	1.847270	0.444161
2022-01-05	1.903297	0.829506	-0.492042	1.245799
2022-01-11	0.593017	1.434123	1.115778	-0.751717
2022-01-13	0.810533	2.313326	-0.747402	-1.227251
2022-01-20	0.297953	0.560134	-0.049372	0.240628

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

```
[ ]:
```

	A	B	C	D
2022-01-01	NaN	0.666944	1.255174	0.620168
2022-01-02	0.679854	0.704714	1.847270	0.444161
2022-01-03	NaN	0.095673	NaN	NaN
2022-01-04	NaN	NaN	0.716966	NaN
2022-01-05	1.903297	0.829506	NaN	1.245799
2022-01-06	NaN	NaN	1.065108	NaN
2022-01-07	NaN	NaN	1.137580	NaN
2022-01-08	NaN	0.294438	1.343623	0.301845
2022-01-09	NaN	NaN	0.090041	0.954330
2022-01-10	NaN	0.941632	0.801956	0.756375
2022-01-11	0.593017	1.434123	1.115778	NaN
2022-01-12	NaN	1.241760	NaN	NaN
2022-01-13	0.810533	2.313326	NaN	NaN

2022-01-14	NaN	0.356303	0.430054	1.050609
2022-01-15	NaN	NaN	2.788002	0.056081
2022-01-16	NaN	NaN	1.490171	NaN
2022-01-17	NaN	0.562094	0.817439	NaN
2022-01-18	NaN	2.166893	NaN	NaN
2022-01-19	NaN	NaN	NaN	NaN
2022-01-20	0.297953	0.560134	NaN	0.240628

```
[ ]: df2 = df.copy()
df2
```

```
[ ]:
      A      B      C      D
2022-01-01 -0.950106  0.666944  1.255174  0.620168
2022-01-02  0.679854  0.704714  1.847270  0.444161
2022-01-03 -0.549860  0.095673 -1.062794 -1.483643
2022-01-04 -0.840756 -1.541808  0.716966 -0.547674
2022-01-05  1.903297  0.829506 -0.492042  1.245799
2022-01-06 -2.087375 -1.134863  1.065108 -0.817172
2022-01-07 -0.333677 -0.782179  1.137580 -1.441844
2022-01-08 -0.588446  0.294438  1.343623  0.301845
2022-01-09 -0.096240 -1.650489  0.090041  0.954330
2022-01-10 -0.649596  0.941632  0.801956  0.756375
2022-01-11  0.593017  1.434123  1.115778 -0.751717
2022-01-12 -0.231228  1.241760 -2.217075 -1.623309
2022-01-13  0.810533  2.313326 -0.747402 -1.227251
2022-01-14 -0.446317  0.356303  0.430054  1.050609
2022-01-15 -1.143818 -0.885201  2.788002  0.056081
2022-01-16 -0.129958 -0.095443  1.490171 -1.012446
2022-01-17 -0.202072  0.562094  0.817439 -1.958470
2022-01-18 -0.002977  2.166893 -0.416769 -0.883259
2022-01-19 -1.469375 -0.034409 -0.486798 -1.216058
2022-01-20  0.297953  0.560134 -0.049372  0.240628
```

```
[ ]: # Adding new column
df2['Babakacolumn'] = ['one', 'one', 'two', 'three', 'four', 'three',
                        'one', 'one', 'two', 'three', 'four', 'three',
                        'one', 'one', 'two', 'three', 'four', 'three', 'four', 'three']
df2
```

```
[ ]:
      A      B      C      D Babakacolumn
2022-01-01 -0.950106  0.666944  1.255174  0.620168      one
2022-01-02  0.679854  0.704714  1.847270  0.444161      one
2022-01-03 -0.549860  0.095673 -1.062794 -1.483643      two
2022-01-04 -0.840756 -1.541808  0.716966 -0.547674      three
2022-01-05  1.903297  0.829506 -0.492042  1.245799      four
2022-01-06 -2.087375 -1.134863  1.065108 -0.817172      three
2022-01-07 -0.333677 -0.782179  1.137580 -1.441844      one
```

2022-01-08	-0.588446	0.294438	1.343623	0.301845	one
2022-01-09	-0.096240	-1.650489	0.090041	0.954330	two
2022-01-10	-0.649596	0.941632	0.801956	0.756375	three
2022-01-11	0.593017	1.434123	1.115778	-0.751717	four
2022-01-12	-0.231228	1.241760	-2.217075	-1.623309	three
2022-01-13	0.810533	2.313326	-0.747402	-1.227251	one
2022-01-14	-0.446317	0.356303	0.430054	1.050609	one
2022-01-15	-1.143818	-0.885201	2.788002	0.056081	two
2022-01-16	-0.129958	-0.095443	1.490171	-1.012446	three
2022-01-17	-0.202072	0.562094	0.817439	-1.958470	four
2022-01-18	-0.002977	2.166893	-0.416769	-0.883259	three
2022-01-19	-1.469375	-0.034409	-0.486798	-1.216058	four
2022-01-20	0.297953	0.560134	-0.049372	0.240628	three

```
[ ]: df2['new'] = [1,2,3,4,5,1,2,3,4,5,1,2,3,4,5,1,2,3,4,5]
df2
```

```
[ ]:
      A      B      C      D Babakacolumn new
2022-01-01 -0.950106  0.666944  1.255174  0.620168      one      1
2022-01-02  0.679854  0.704714  1.847270  0.444161      one      2
2022-01-03 -0.549860  0.095673 -1.062794 -1.483643      two      3
2022-01-04 -0.840756 -1.541808  0.716966 -0.547674     three      4
2022-01-05  1.903297  0.829506 -0.492042  1.245799      four      5
2022-01-06 -2.087375 -1.134863  1.065108 -0.817172     three      1
2022-01-07 -0.333677 -0.782179  1.137580 -1.441844      one      2
2022-01-08 -0.588446  0.294438  1.343623  0.301845      one      3
2022-01-09 -0.096240 -1.650489  0.090041  0.954330      two      4
2022-01-10 -0.649596  0.941632  0.801956  0.756375     three      5
2022-01-11  0.593017  1.434123  1.115778 -0.751717      four      1
2022-01-12 -0.231228  1.241760 -2.217075 -1.623309     three      2
2022-01-13  0.810533  2.313326 -0.747402 -1.227251      one      3
2022-01-14 -0.446317  0.356303  0.430054  1.050609      one      4
2022-01-15 -1.143818 -0.885201  2.788002  0.056081      two      5
2022-01-16 -0.129958 -0.095443  1.490171 -1.012446     three      1
2022-01-17 -0.202072  0.562094  0.817439 -1.958470      four      2
2022-01-18 -0.002977  2.166893 -0.416769 -0.883259     three      3
2022-01-19 -1.469375 -0.034409 -0.486798 -1.216058      four      4
2022-01-20  0.297953  0.560134 -0.049372  0.240628     three      5
```

```
[ ]: # Getting first four columns
df2= df2.iloc[:, 0:4]
df2
```

```
[ ]:
      A      B      C      D
2022-01-01 -0.950106  0.666944  1.255174  0.620168
2022-01-02  0.679854  0.704714  1.847270  0.444161
2022-01-03 -0.549860  0.095673 -1.062794 -1.483643
```

```

2022-01-04 -0.840756 -1.541808  0.716966 -0.547674
2022-01-05  1.903297  0.829506 -0.492042  1.245799
2022-01-06 -2.087375 -1.134863  1.065108 -0.817172
2022-01-07 -0.333677 -0.782179  1.137580 -1.441844
2022-01-08 -0.588446  0.294438  1.343623  0.301845
2022-01-09 -0.096240 -1.650489  0.090041  0.954330
2022-01-10 -0.649596  0.941632  0.801956  0.756375
2022-01-11  0.593017  1.434123  1.115778 -0.751717
2022-01-12 -0.231228  1.241760 -2.217075 -1.623309
2022-01-13  0.810533  2.313326 -0.747402 -1.227251
2022-01-14 -0.446317  0.356303  0.430054  1.050609
2022-01-15 -1.143818 -0.885201  2.788002  0.056081
2022-01-16 -0.129958 -0.095443  1.490171 -1.012446
2022-01-17 -0.202072  0.562094  0.817439 -1.958470
2022-01-18 -0.002977  2.166893 -0.416769 -0.883259
2022-01-19 -1.469375 -0.034409 -0.486798 -1.216058
2022-01-20  0.297953  0.560134 -0.049372  0.240628

```

```
[ ]: df3 = df[df['A'] > 0]
df3
```

```
[ ]:
      A      B      C      D
2022-01-02  0.679854  0.704714  1.847270  0.444161
2022-01-05  1.903297  0.829506 -0.492042  1.245799
2022-01-11  0.593017  1.434123  1.115778 -0.751717
2022-01-13  0.810533  2.313326 -0.747402 -1.227251
2022-01-20  0.297953  0.560134 -0.049372  0.240628

```

```
[ ]: df3['mean'] = [(df3.iloc[0]).mean(),(df3.iloc[1]).mean(),(df3.iloc[2]).
↪mean(),(df3.iloc[3]).mean(),(df3.iloc[4]).mean())]
df3
```

C:\Users\Waleed\AppData\Local\Temp\ipykernel_9912\3693726348.py:1:

SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
df3['mean'] = [(df3.iloc[0]).mean(),(df3.iloc[1]).mean(),(df3.iloc[2]).mean(),
(df3.iloc[3]).mean(),(df3.iloc[4]).mean())]
```

```
[ ]:
      A      B      C      D      mean
2022-01-02  0.679854  0.704714  1.847270  0.444161  0.918999
2022-01-05  1.903297  0.829506 -0.492042  1.245799  0.871640
2022-01-11  0.593017  1.434123  1.115778 -0.751717  0.597800
2022-01-13  0.810533  2.313326 -0.747402 -1.227251  0.287302
2022-01-20  0.297953  0.560134 -0.049372  0.240628  0.262336

```


0.16.1 Pandas Case Study

import save and analyze Data using Pandas (Titanic Dataset)

```
[ ]: import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

```
[ ]: # import titanic (kashti) dataset
df = sns.load_dataset('titanic')
df.head()
```

```
[ ]:      survived  pclass    sex  age  sibsp  parch    fare embarked  class \
0           0         3   male  22.0     1     0   7.2500         S   Third
1           1         1  female  38.0     1     0  71.2833         C   First
2           1         3  female  26.0     0     0   7.9250         S   Third
3           1         1  female  35.0     1     0  53.1000         S   First
4           0         3   male  35.0     0     0   8.0500         S   Third

      who  adult_male  deck  embark_town  alive  alone
0   man           True  NaN  Southampton    no  False
1 woman          False    C   Cherbourg   yes  False
2 woman          False  NaN  Southampton   yes   True
3 woman          False    C   Southampton   yes  False
4   man           True  NaN  Southampton    no   True
```

```
[ ]: # Saving dataset as csv file
df.to_csv('titanic.csv')
```

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

```
[ ]: (891, 15)
```

```
[ ]: # Basic Statistics or Summary
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
```

```
[ ]: # Dropping few columns and make a new dataset
df1 = df.drop(['deck', 'alone'], axis=1)
df1.head()
```

```
[ ]:      survived  pclass      sex  age  sibsp  parch      fare embarked  class \
0           0        3    male  22.0     1     0   7.2500          S   Third
1           1        1  female  38.0     1     0  71.2833          C   First
2           1        3  female  26.0     0     0   7.9250          S   Third
3           1        1  female  35.0     1     0  53.1000          S   First
4           0        3    male  35.0     0     0   8.0500          S   Third

      who  adult_male  embark_town  alive
0    man          True  Southampton    no
1  woman         False   Cherbourg   yes
2  woman         False  Southampton   yes
3  woman         False  Southampton   yes
4    man          True  Southampton    no
```

```
[ ]: df1.shape
```

```
[ ]: (891, 13)
```

```
[ ]: df1.mean()
```

C:\Users\Waleed\AppData\Local\Temp\ipykernel_9912\2053335143.py:1:
FutureWarning: Dropping of nuisance columns in DataFrame reductions (with
'numeric_only=None') is deprecated; in a future version this will raise
TypeError. Select only valid columns before calling the reduction.
df1.mean()

```
[ ]: survived      0.383838
pclass           2.308642
age             29.699118
sibsp           0.523008
parch           0.381594
fare           32.204208
adult_male      0.602694
dtype: float64
```

```
[ ]: # groupby
df1.groupby(['sex', 'class']).mean()
```

```
[ ]:      survived  pclass      age  sibsp  parch      fare \
sex  class
female First  0.968085    1.0  34.611765  0.553191  0.457447  106.125798
      Second  0.921053    2.0  28.722973  0.486842  0.605263   21.970121
      Third  0.500000    3.0  21.750000  0.895833  0.798611   16.118810
male   First  0.368852    1.0  41.281386  0.311475  0.278689   67.226127
```

Second	0.157407	2.0	30.740707	0.342593	0.222222	19.741782
Third	0.135447	3.0	26.507589	0.498559	0.224784	12.661633

adult_male		
sex	class	
female	First	0.000000
	Second	0.000000
	Third	0.000000
male	First	0.975410
	Second	0.916667
	Third	0.919308

```
[ ]: df1.value_counts(['survived'])
```

```
[ ]: survived
0          549
1          342
dtype: int64
```

```
[ ]: df1.groupby(['sex']).mean()
```

```
[ ]:      survived    pclass      age    sibsp    parch      fare \
sex
female  0.742038    2.159236  27.915709  0.694268  0.649682  44.479818
male    0.188908    2.389948  30.726645  0.429809  0.235702  25.523893

      adult_male
sex
female    0.000000
male      0.930676
```

```
[ ]: df1.groupby(['sex', 'class']).mean()
```

```
[ ]:      survived    pclass      age    sibsp    parch      fare \
sex  class
female First    0.968085      1.0  34.611765  0.553191  0.457447  106.125798
      Second    0.921053      2.0  28.722973  0.486842  0.605263   21.970121
      Third     0.500000      3.0  21.750000  0.895833  0.798611   16.118810
male   First    0.368852      1.0  41.281386  0.311475  0.278689   67.226127
      Second    0.157407      2.0  30.740707  0.342593  0.222222   19.741782
      Third     0.135447      3.0  26.507589  0.498559  0.224784   12.661633

      adult_male
sex  class
female First    0.000000
      Second    0.000000
      Third     0.000000
```

male	First	0.975410
	Second	0.916667
	Third	0.919308

```
[ ]: # Under 18 years
df1[df1['age'] < 18].groupby(['sex', 'class']).mean()
```

```
[ ]:
      survived  pclass      age  sibsp  parch      fare \
sex  class
female First    0.875000     1.0  14.125000  0.500000  0.875000  104.083337
      Second    1.000000     2.0   8.333333  0.583333  1.083333   26.241667
      Third    0.542857     3.0   8.428571  1.571429  1.057143   18.727977
male   First    1.000000     1.0   8.230000  0.500000  2.000000  116.072900
      Second    0.818182     2.0   4.757273  0.727273  1.000000   25.659473
      Third    0.232558     3.0   9.963256  2.069767  1.000000   22.752523
```

```
      adult_male
sex  class
female First    0.000000
      Second    0.000000
      Third    0.000000
male   First    0.250000
      Second    0.181818
      Third    0.348837
```

0.17 Statistics

Statistics is a collection of methods for collecting, displaying, > analyzing and drawing conclusions from data. **Statistics is everywhere:**

- Weather Prediction
- USD Prediction
- ANOVA
- Un employment rate fallen
- etc

Language of Statistics:

- **Average** income in Pakistan
- Highest (**Maximum**) score in cricket match
- 40% (**Percentage**) teachers in Pakistan are female
- Dollar kabhi uper jata hy kabhi neechay (**Variance**)
- Hostels main larkay zaida kharcha kartay hyn (**t-test**)
- Faislabad > Lahore > Karachi had jugtain ke ranking (**ANOVA**)

Types of Data

1. Cross Sectional Data
 - Data collected at one point
2. Time Series Data

- Data Collected over different time points
3. Univariate
 - Data contains a single variable to measure entity e.g:
 1. Plant Height
 4. Multi Variate
 - Data contains > 2 variables to measure something e.g:
 1. Plant Height
 2. Fertilizer Amount
 3. Irrigation Time

Types of Variables

1. Categorical (Nominal)
 1. Binomial (True or False)
 2. Multinomial (multiple choices e.g. how to go to office)
 3. Ordinal Variable: Data ranked or ordered (mery pass kitny phone hain?, categories can be compared)
2. Ratio Data:
 1. Data have a natural zero, Measurement in units and ratios are continuous variable
3. Interval Variable / Data:
 1. ordered and characterized data

Measure of Central Tendency

1. Mean:
 - Average, meaningful for interval and ratio data
 - Outliers: change the mean of a data, therefore median is useful
2. Median:
 - Middle number in a sorted, ascending or descending, list of numbers
3. Mode:
 - The value that occurs most frequently

Measure of Dispersion

1. Dispersion:
 - How much data spread around mean
2. Standard Deviation (std)
 - **Mean** with a **SD** is more useful than only Mean by itself
3. Standard Error (se)
4. Variance
5. Bell Curve

Fundamentals of Visualization

Type of variable depends on the variable type

1. Categorical Variable: Qualitative (No numerical meaning)
 - Counts (plot type)
 - Male vs Female
 - True vs False
 - 0 vs 1

- Yes vs NO
2. Continuous Variable: Quantitative Numerical (mostly represented in numbers)
 - Scatter plot
 - Statistical Proportions (means and their comparison)

0.18 Exploratory Data Analysis (EDA)

0.18.1 Three important steps to keep in mind:

- Understand the data
- Clean the data
- Find a relationship between data

```
[ ]: import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[ ]: ks = sns.load_dataset('titanic')
ks.head()
```

```
[ ]:      survived  pclass    sex   age  sibsp  parch    fare embarked  class \
0          0         3   male  22.0     1     0   7.2500         S   Third
1          1         1  female  38.0     1     0  71.2833         C   First
2          1         3  female  26.0     0     0   7.9250         S   Third
3          1         1  female  35.0     1     0  53.1000         S   First
4          0         3   male  35.0     0     0   8.0500         S   Third
```

```
      who  adult_male  deck  embark_town  alive  alone
0   man         True  NaN  Southampton    no  False
1  woman        False    C   Cherbourg   yes  False
2  woman        False  NaN  Southampton   yes   True
3  woman        False    C   Southampton   yes  False
4   man         True  NaN  Southampton    no   True
```

```
[ ]: ks.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

```

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

```
[ ]: (891, 15)
```

```
[ ]: # Provides details of numeric columns
ks.describe()
```

```
[ ]:
count    survived      pclass      age      sibsp      parch      fare
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

```

```
[ ]: # Unique values in each column
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

```

```
[ ]: # Column names
ks.columns
```

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

```
[ ]: # unique values in specific column
ks['sex'].unique()
```

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

If you wanted to get all unique values for one column and then the second column use argument 'K' to the ravel() function. The argument 'K' tells the method to flatten the array in the order of the elements.

```
[ ]: # Using pandas.unique() to get unique values in multiple columns
df = pd.unique(ks[['sex', 'class']].values.ravel('k'))
print(df)
```

```
['male' 'female' 'Third' 'First' 'Second']
```

```
[ ]: # Use numpy.unique() to get unique values in multiple columns
column_values = ks[['sex', 'class']].values
df2 = np.unique(column_values)
print(df2)
```

```
['First' 'Second' 'Third' 'female' 'male']
```

Cleaning and Filtering Data

```
[ ]: # find missing values inside
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

```
[ ]: # Removing missing values column (cleaning data)
ks_clean = ks.drop(['deck'], axis=1)
print('ks.shape: ', ks.shape)
print('ks_clean.shape: ', ks_clean.shape)
```

ks.shape: (891, 15)

ks_clean.shape: (891, 14)

```
[ ]: # After removing deck column finding missing values again
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
```

```
[ ]: # Removing all null values
ks_clean = ks_clean.dropna()
print('ks_clean.shape: ', ks_clean.shape)
ks_clean.isnull().sum()
```

ks_clean.shape: (712, 14)

```
[ ]: 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
```

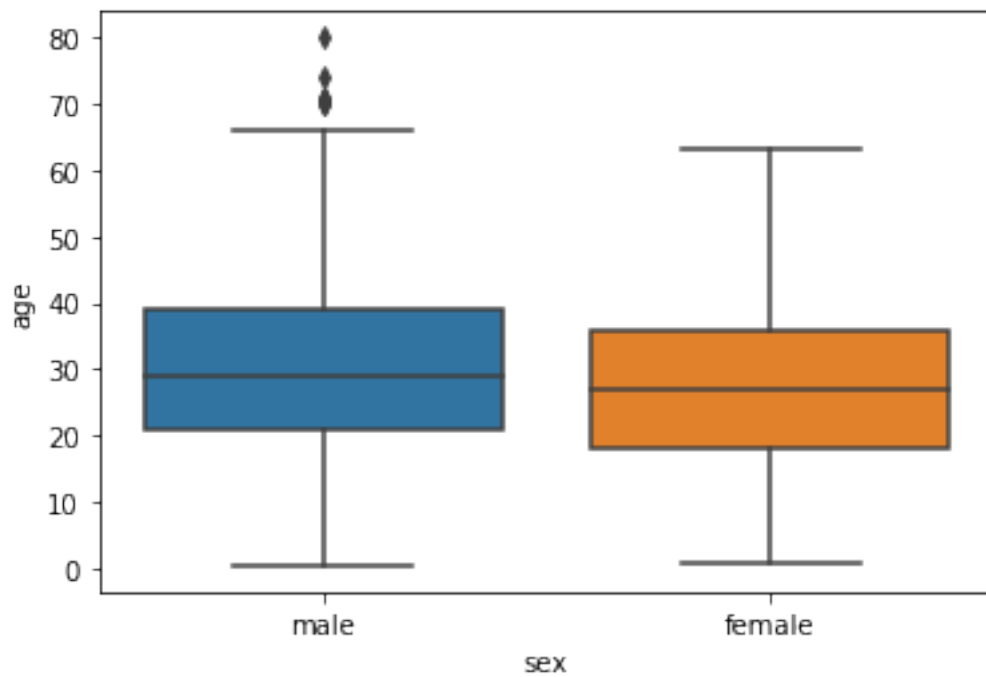
```
alive      0
alone      0
dtype: int64
```

```
[ ]: # Counting Values in specific column
ks_clean['sex'].value_counts()
```

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

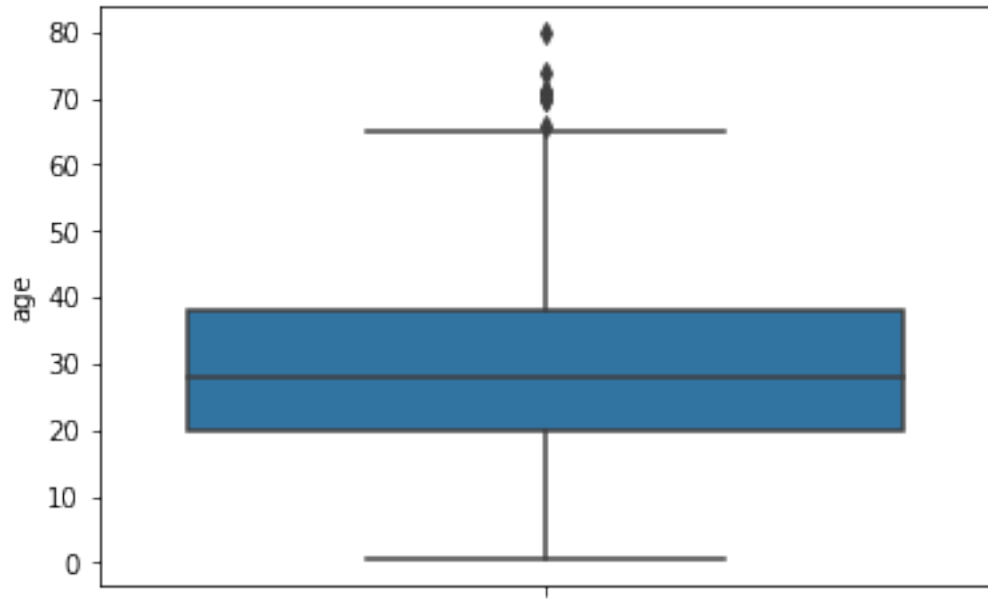
```
[ ]: # Finding outliers
sns.boxplot(x='sex', y='age', data=ks_clean)
```

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



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

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

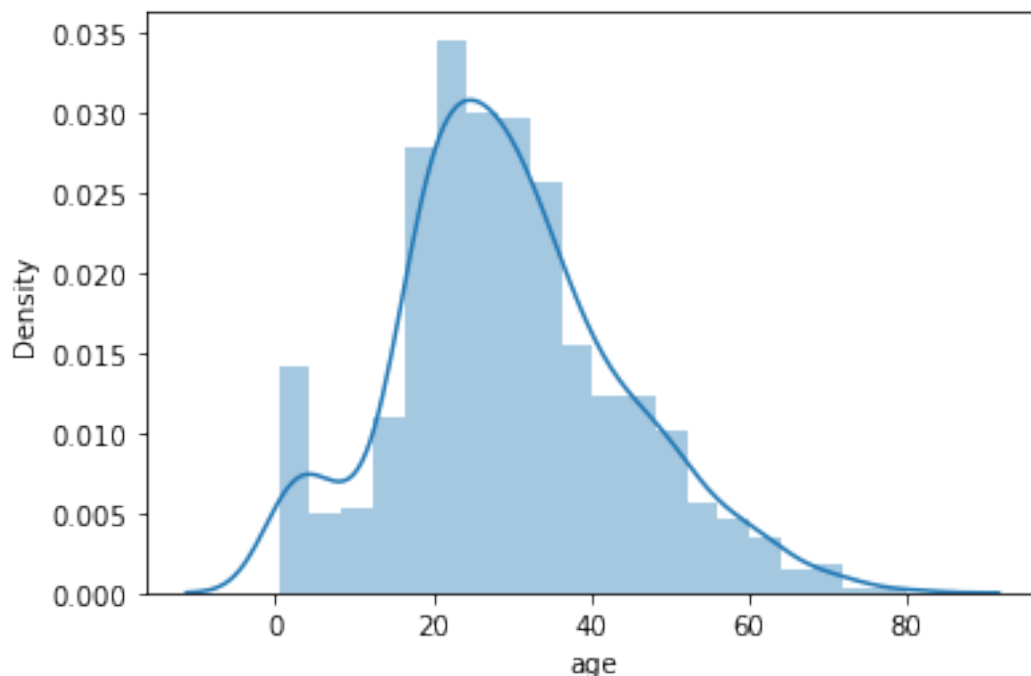


```
[ ]: # Data Distribution
sns.distplot(ks_clean['age'])
```

```
c:\Users\Waleed\anaconda3\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).
```

```
warnings.warn(msg, FutureWarning)
```

```
[ ]: <AxesSubplot:xlabel='age', ylabel='Density'>
```



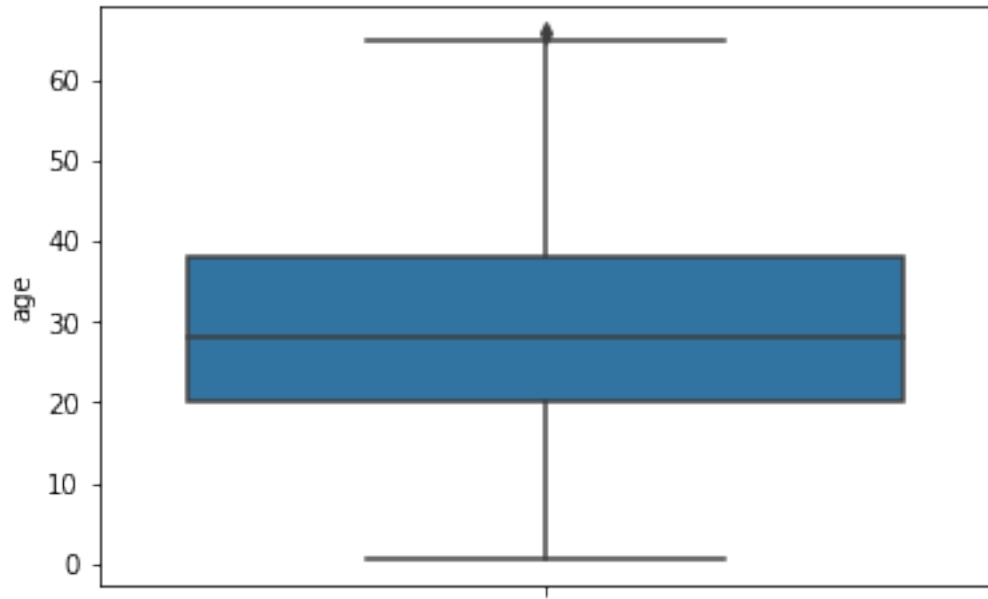
```
[ ]: # Removing Outliers
ks_clean = ks_clean[ks_clean['age'] < 68]
ks_clean.head()
```

```
[ ]:   survived  pclass    sex  age  sibsp  parch    fare embarked  class \
0         0        3   male  22.0     1     0    7.2500         S   Third
1         1        1  female  38.0     1     0   71.2833         C   First
2         1        3  female  26.0     0     0    7.9250         S   Third
3         1        1  female  35.0     1     0   53.1000         S   First
4         0        3   male  35.0     0     0    8.0500         S   Third

      who  adult_male  embark_town  alive  alone
0   man         True  Southampton    no  False
1 woman        False   Cherbourg   yes  False
2 woman        False  Southampton   yes   True
3 woman        False  Southampton   yes  False
4   man         True  Southampton    no   True
```

```
[ ]: # After filtering outliers
sns.boxplot(y='age', data=ks_clean)
```

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



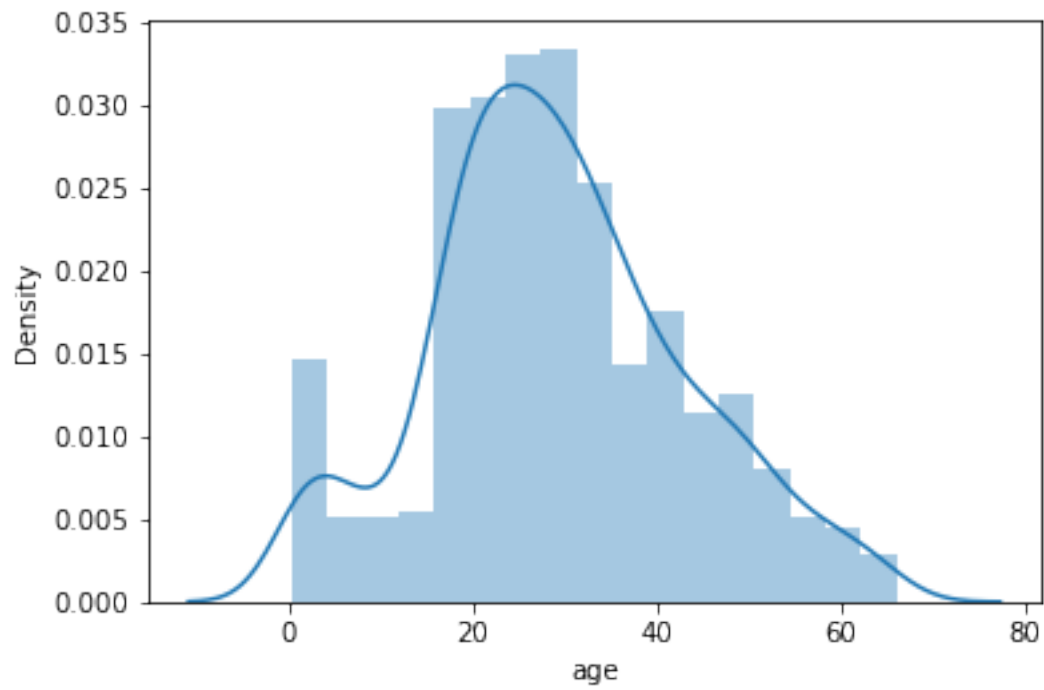
Looks Much Better

```
[ ]: sns.distplot(ks_clean['age'])
```

```
c:\Users\Waleed\anaconda3\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).
```

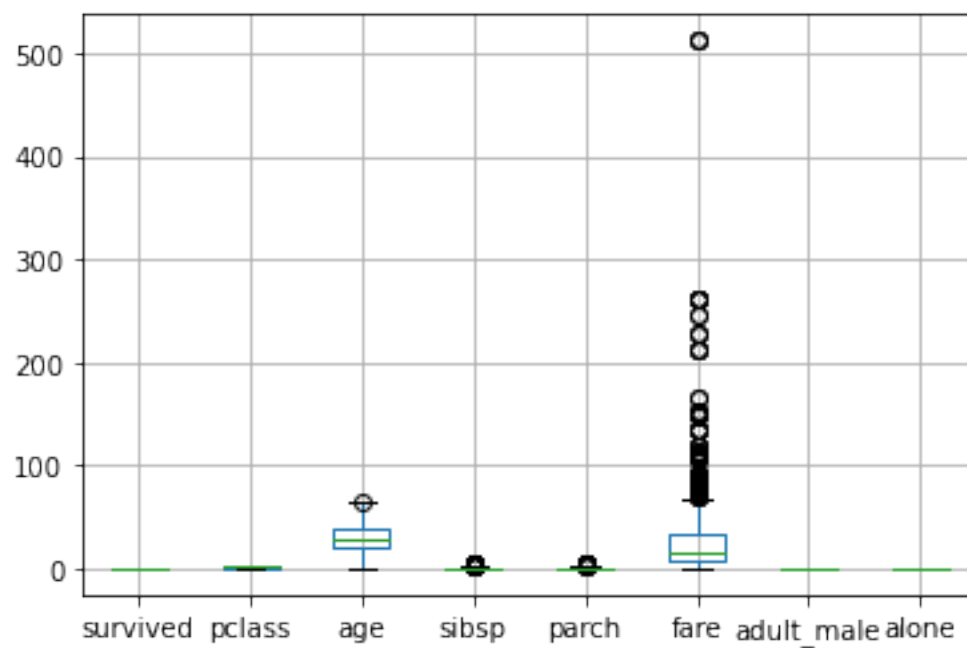
```
warnings.warn(msg, FutureWarning)
```

```
[ ]: <AxesSubplot:xlabel='age', ylabel='Density'>
```



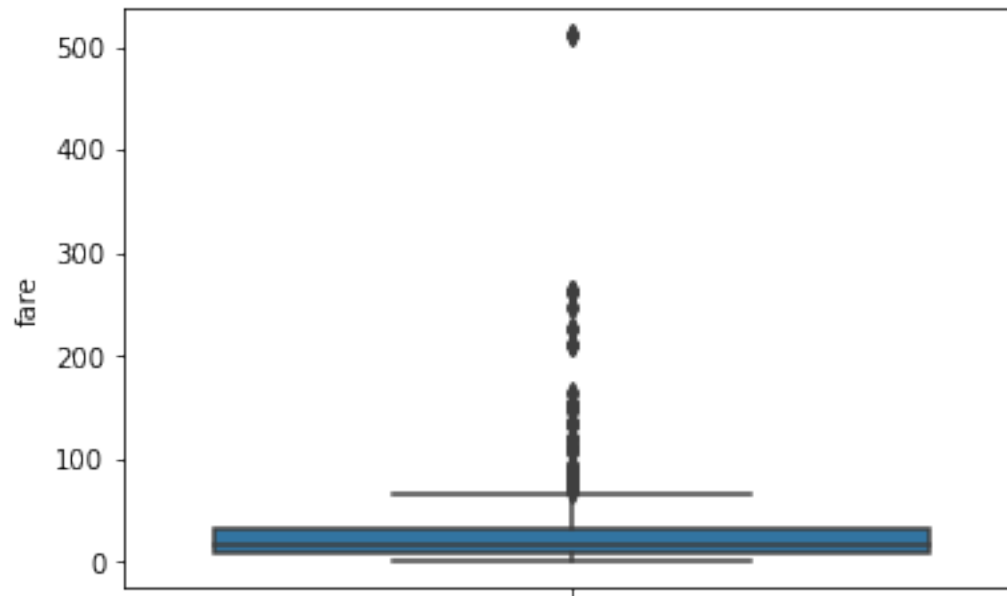
```
[ ]: # Whole Data BoxPlot
ks_clean.boxplot()
```

```
[ ]: <AxesSubplot:>
```



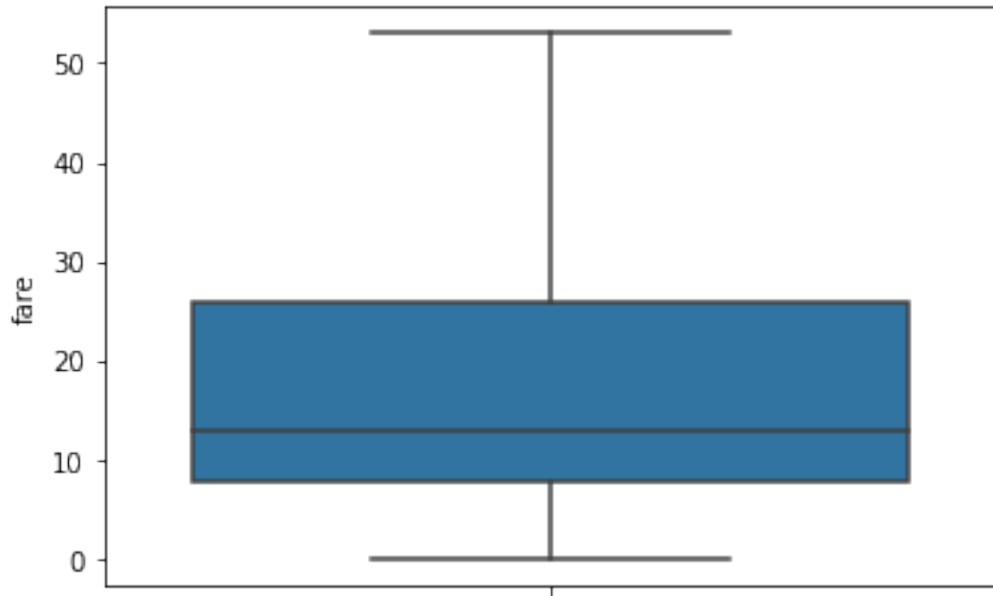
```
[ ]: sns.boxplot(y='fare', data=ks_clean)
```

```
[ ]: <AxesSubplot:ylabel='fare'>
```



```
[ ]: # cleaning fare column  
ks_clean = ks_clean[ks_clean['fare'] < 55]  
sns.boxplot(y='fare', data=ks_clean)
```

```
[ ]: <AxesSubplot:ylabel='fare'>
```

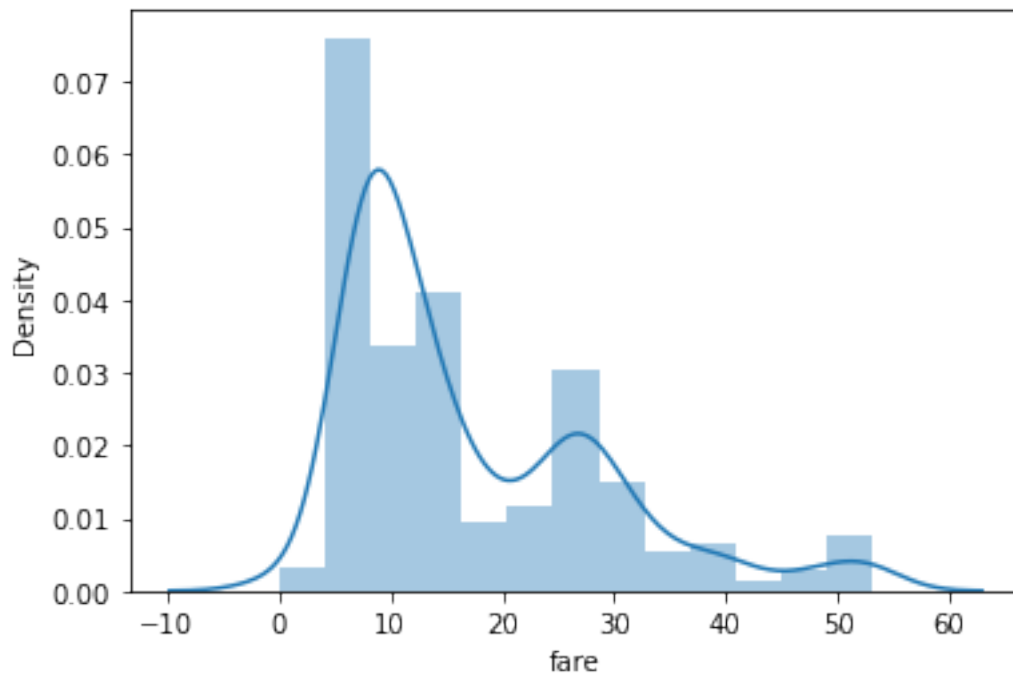


```
[ ]: sns.distplot(ks_clean['fare'])
```

```
c:\Users\Waleed\anaconda3\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).
```

```
warnings.warn(msg, FutureWarning)
```

```
[ ]: <AxesSubplot:xlabel='fare', ylabel='Density'>
```

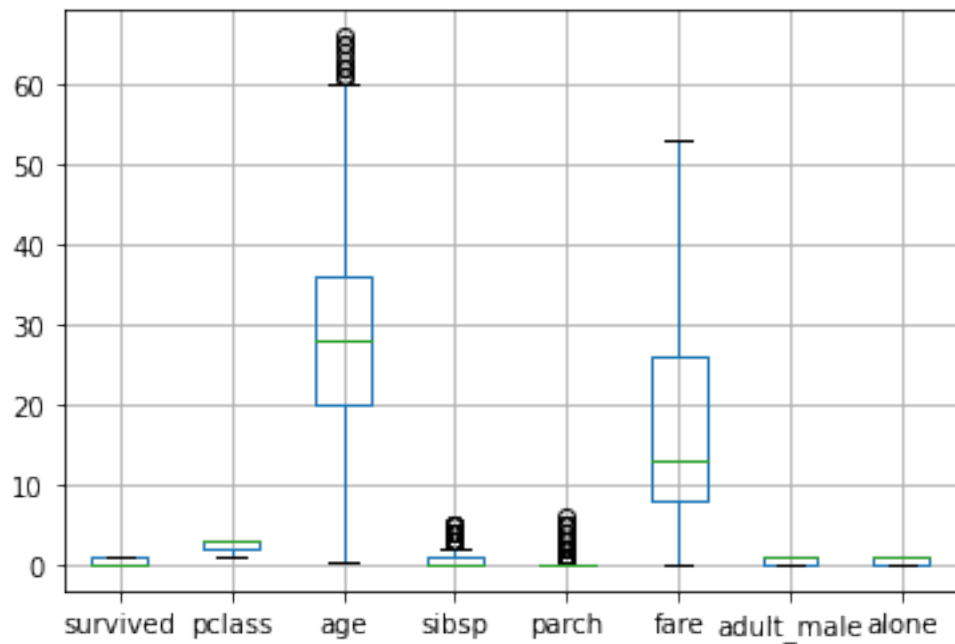



Data is not normally distributed, log transformation can be used to resolve the issue.

```
[ ]: # log transformation
ks_clean['fare_log'] = np.log(ks_clean['fare'])
```

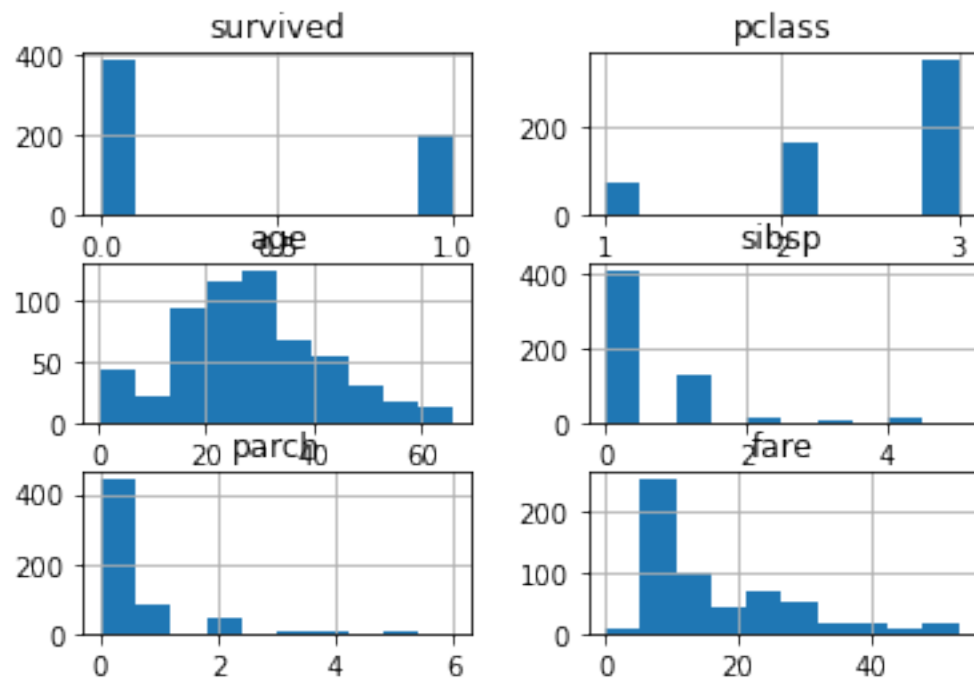
```
[ ]: ks_clean.boxplot()
```

```
[ ]: <AxesSubplot:>
```



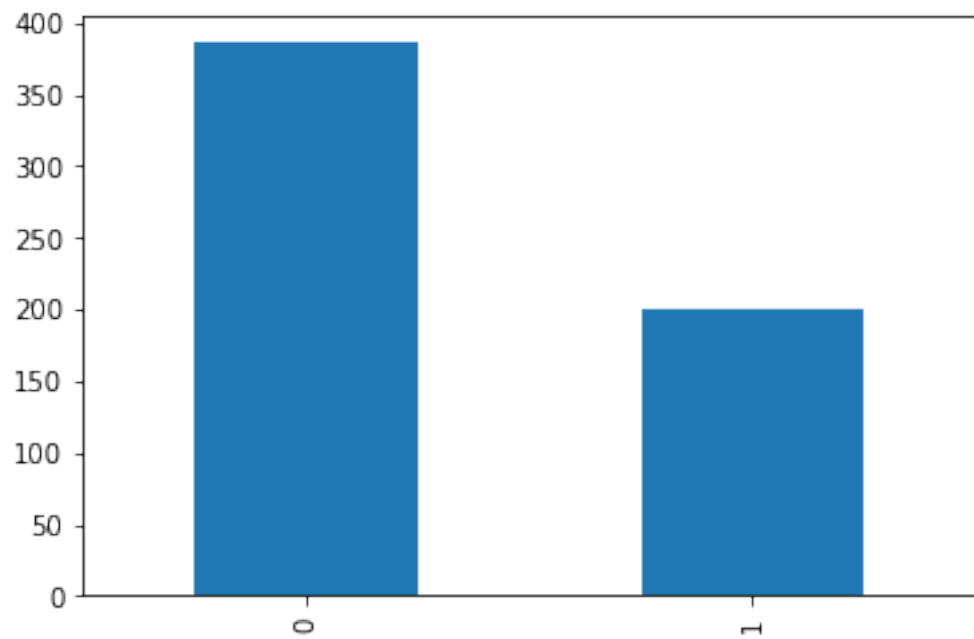
```
[ ]: # Complete data histogram
ks_clean.hist()
```

```
[ ]: 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)
```



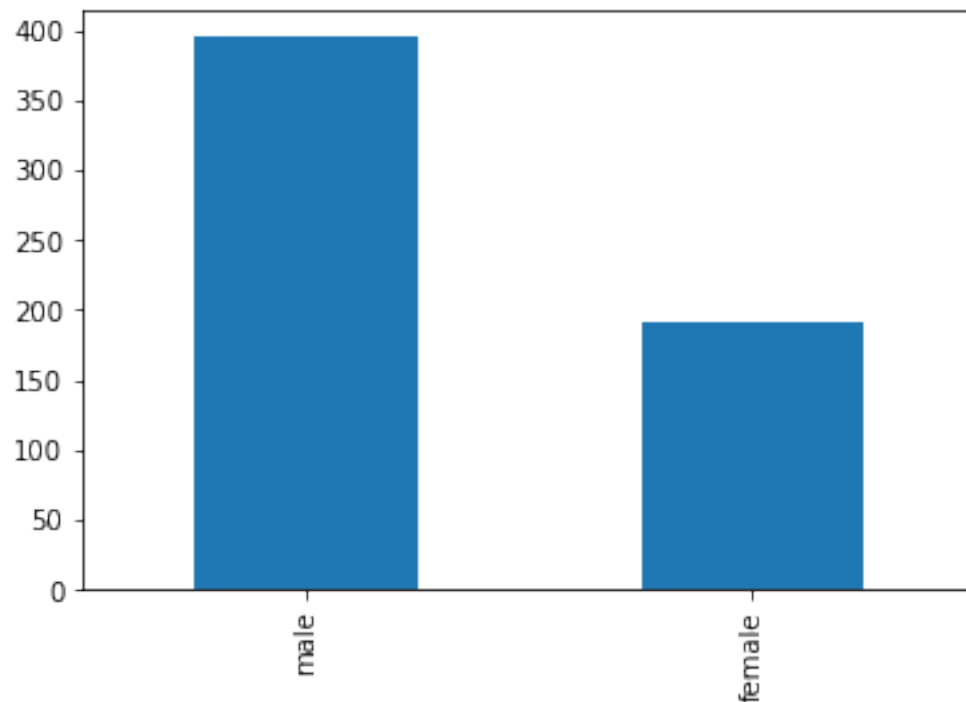
```
[ ]: pd.value_counts(ks_clean['survived']).plot.bar()
```

```
[ ]: <AxesSubplot:>
```



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

```
[ ]: <AxesSubplot:>
```



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

```
[ ]:
```

		survived	pclass	age	sibsp	parch	fare \
female	First	0.941176	1.0	37.058824	0.411765	0.352941	38.962506
	Second	0.916667	2.0	28.520833	0.486111	0.583333	20.755267
	Third	0.460784	3.0	21.750000	0.823529	0.950980	15.875369
male	First	0.381818	1.0	42.772727	0.145455	0.036364	32.260680
	Second	0.161290	2.0	30.723978	0.333333	0.258065	18.410753
	Third	0.141700	3.0	26.088745	0.502024	0.263158	11.480379

		adult_male	alone
female	First	0.000000	0.470588
	Second	0.000000	0.416667
	Third	0.000000	0.372549
male	First	1.000000	0.836364
	Second	0.903226	0.645161
	Third	0.886640	0.732794

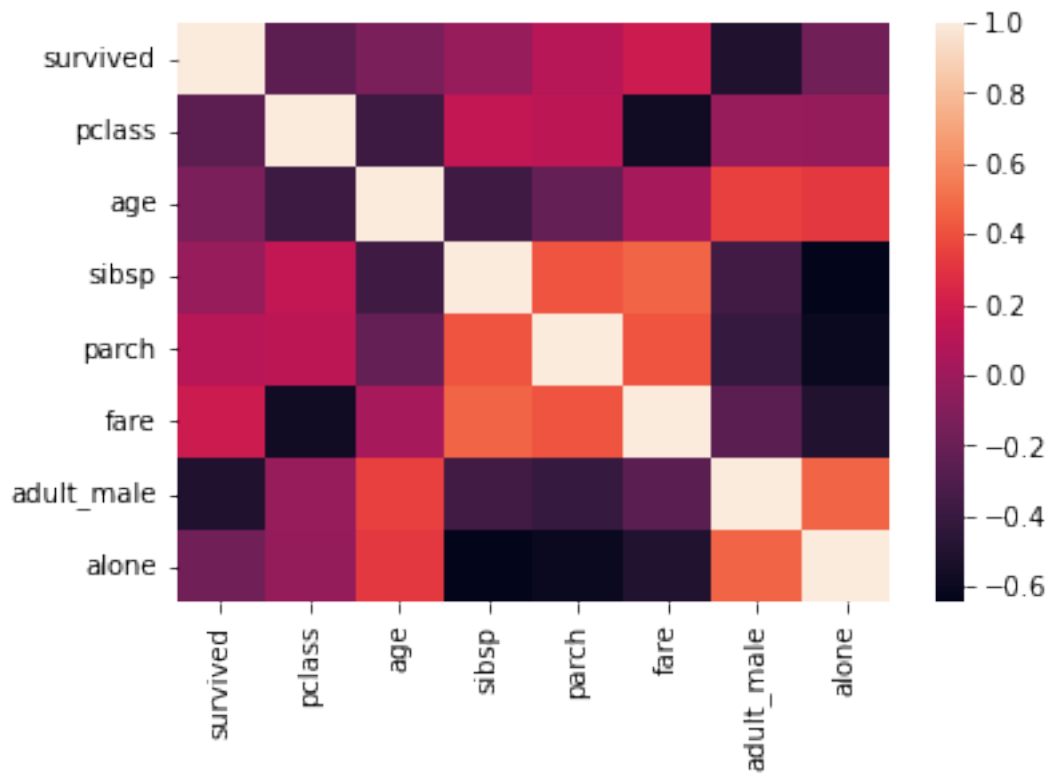
```
[ ]: # correlation matrix 1 shows positive whereas neg value shows neg correlation
ks_clean.corr()
```

```
[ ]:
survived    pclass    age    sibsp    parch    fare \
survived    1.000000 -0.253421 -0.137448 -0.028300 0.098433 0.176168
pclass      -0.253421 1.000000 -0.380304 0.140866 0.120047 -0.573454
age         -0.137448 -0.380304 1.000000 -0.372958 -0.223097 0.033772
sibsp       -0.028300 0.140866 -0.372958 1.000000 0.411134 0.464386
parch       0.098433 0.120047 -0.223097 0.411134 1.000000 0.411263
fare        0.176168 -0.573454 0.033772 0.464386 0.411263 1.000000
adult_male -0.510726 -0.025996 0.341861 -0.362488 -0.412959 -0.260964
alone       -0.177098 -0.032915 0.312785 -0.648029 -0.599167 -0.503869

          adult_male    alone
survived    -0.510726 -0.177098
pclass      -0.025996 -0.032915
age          0.341861 0.312785
sibsp       -0.362488 -0.648029
parch       -0.412959 -0.599167
fare        -0.260964 -0.503869
adult_male   1.000000 0.463082
alone        0.463082 1.000000
```

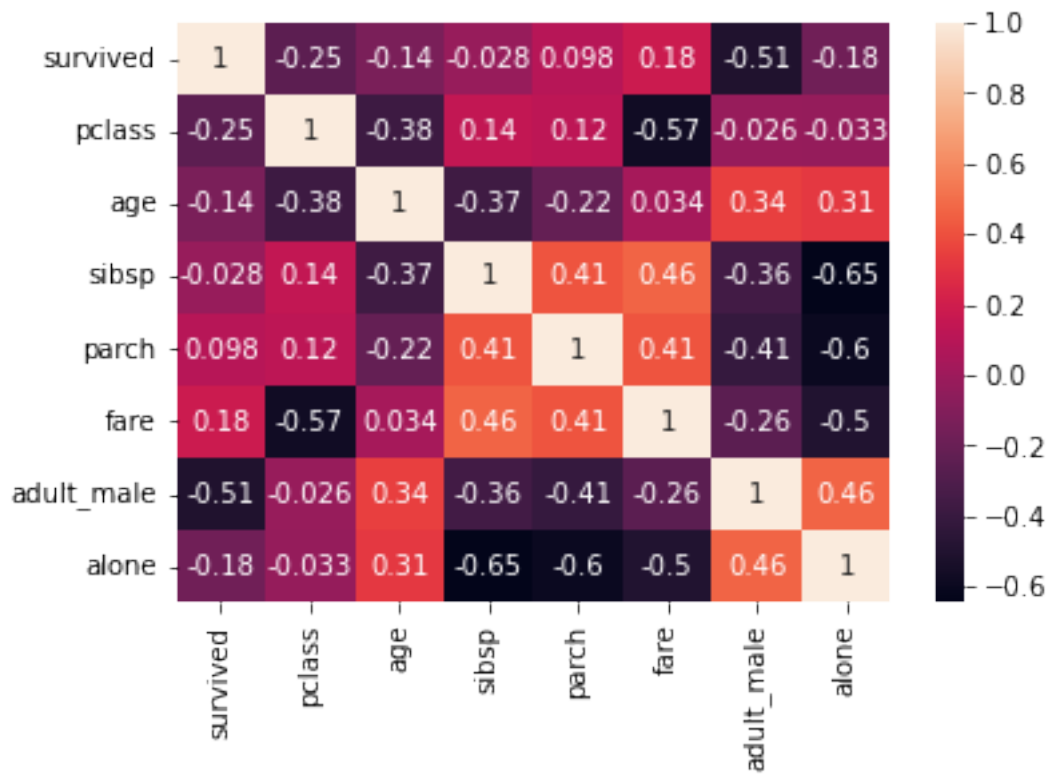
```
[ ]: corr_ks_clean = ks_clean.corr()
sns.heatmap(corr_ks_clean)
```

```
[ ]: <AxesSubplot:>
```



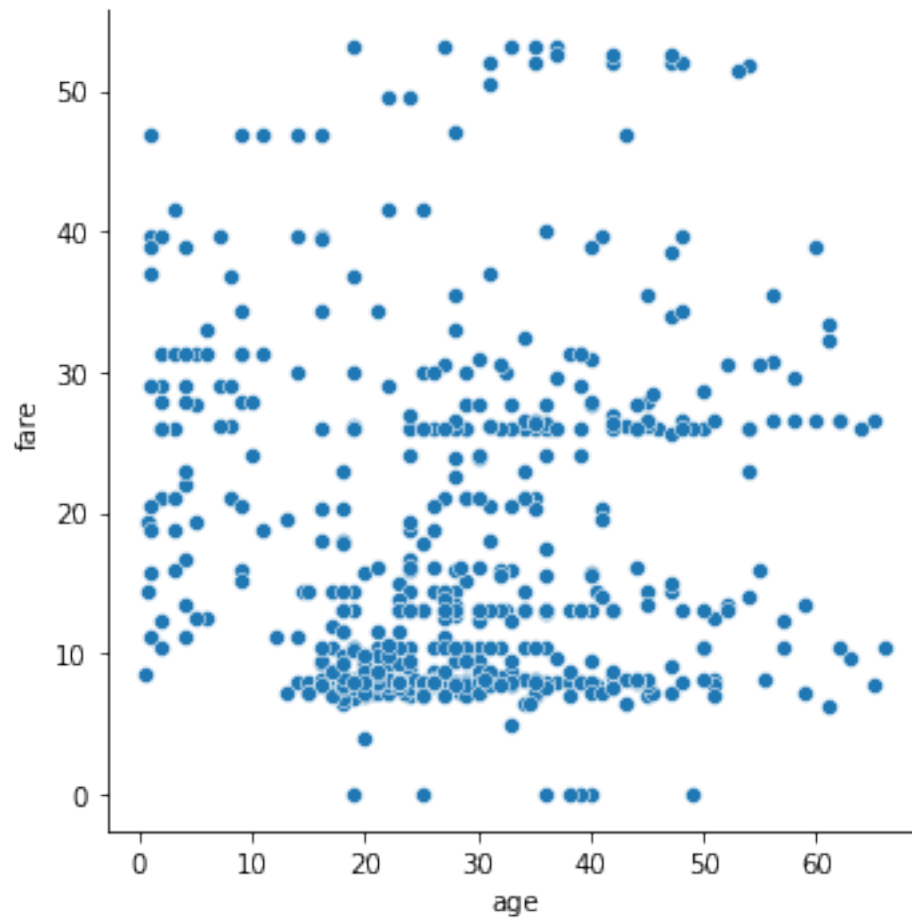
```
[ ]: sns.heatmap(corr_ks_clean, annot=True)
```

```
[ ]: <AxesSubplot:>
```



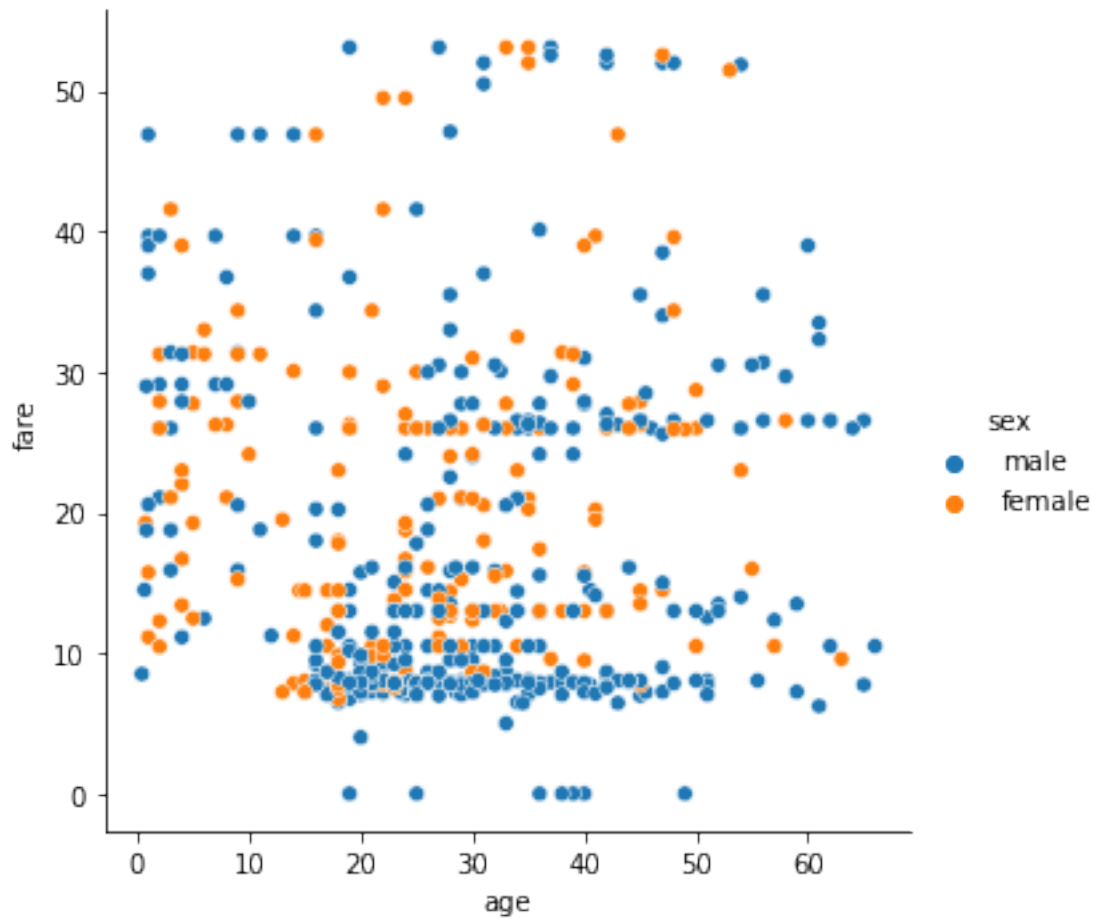
```
[ ]: sns.relplot(x= 'age', y='fare', data=ks_clean)
```

```
[ ]: <seaborn.axisgrid.FacetGrid at 0x1fdc67e8190>
```



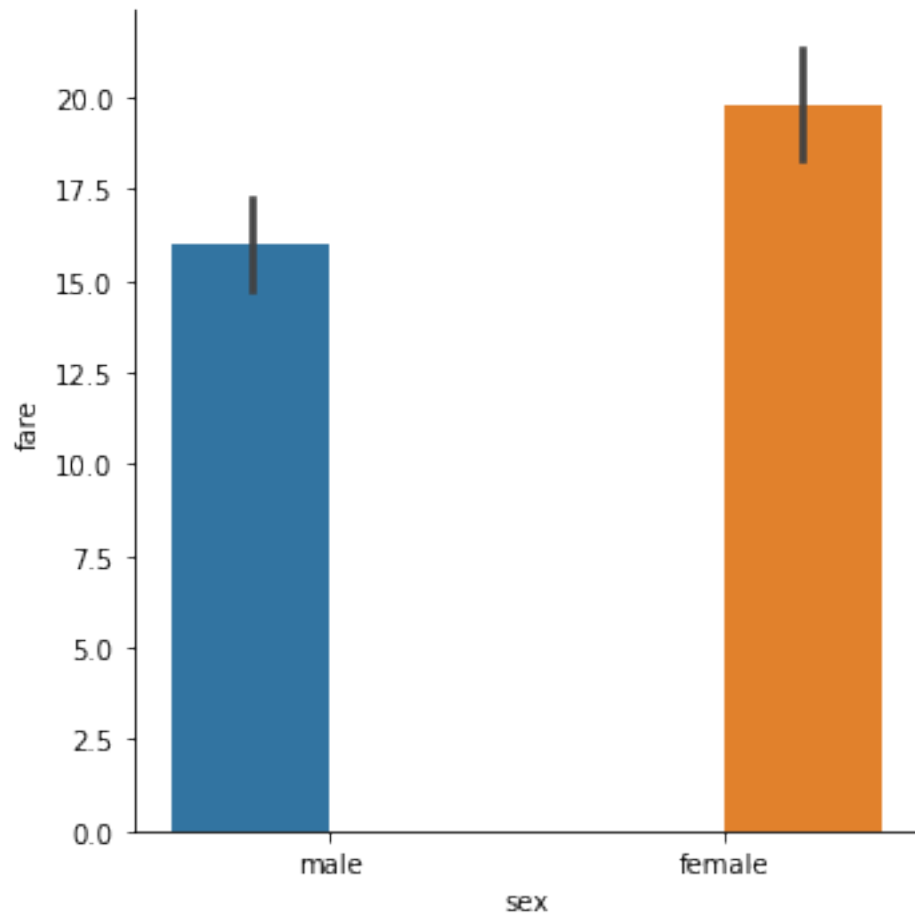
```
[ ]: # Grouping
sns.relplot(x= 'age', y='fare',hue='sex', data=ks_clean)
```

```
[ ]: <seaborn.axisgrid.FacetGrid at 0x1fdc677a3a0>
```

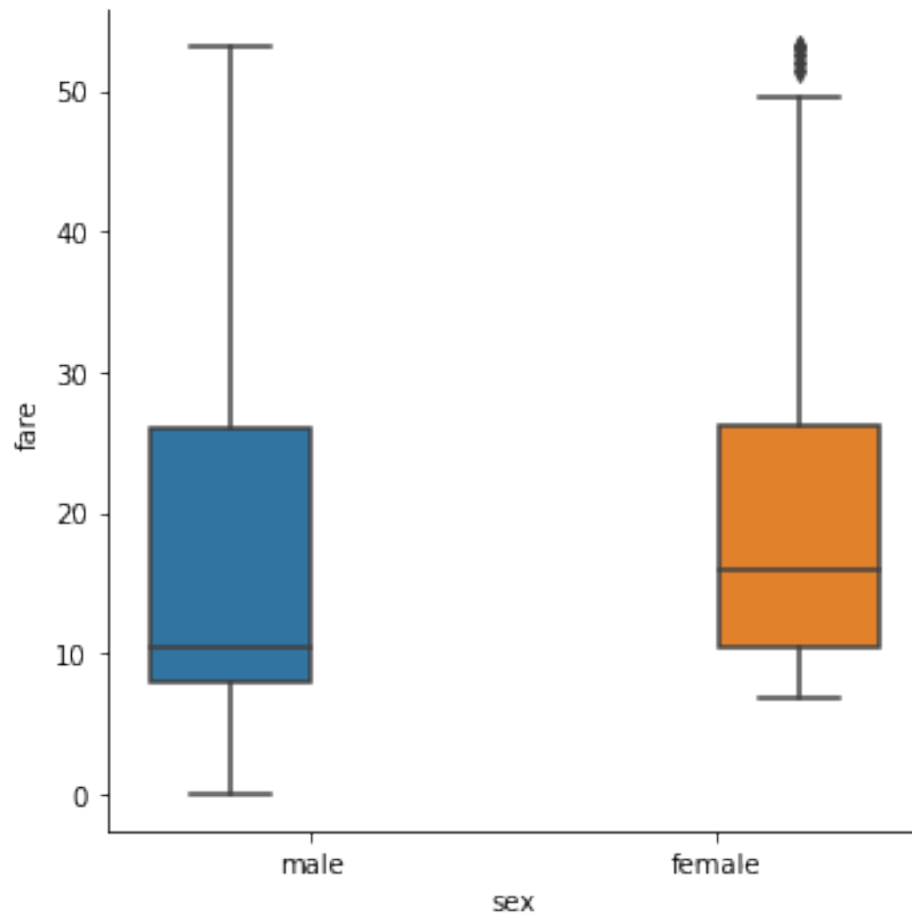
```
[ ]: # Category plot
sns.catplot(x= 'sex', y='fare',hue='sex', data=ks_clean, kind='bar')
```

```
[ ]: <seaborn.axisgrid.FacetGrid at 0x1fdc79c2d90>
```



```
[ ]: sns.catplot(x= 'sex', y='fare',hue='sex', data=ks_clean, kind='box')
```

```
[ ]: <seaborn.axisgrid.FacetGrid at 0x1fdc7ad2dc0>
```



0.19 Data Wrangling

```
[ ]: # Importing Required Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[ ]: # Importing Dataset
kashti = sns.load_dataset('titanic')
kashti.head()
```

```
[ ]:   survived  pclass    sex  age  sibsp  parch   fare embarked  class \
0         0      3   male  22.0     1     0   7.2500         S   Third
1         1      1  female  38.0     1     0  71.2833         C   First
2         1      3  female  26.0     0     0   7.9250         S   Third
3         1      1  female  35.0     1     0  53.1000         S   First
4         0      3   male  35.0     0     0   8.0500         S   Third
```

	who	adult_male	deck	embark_town	alive	alone
0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

```
[ ]: # Simple Operations (Math Operators)
# Doing simple math operations on numeric value columns

(kashti['age'] * 6).head(10)
```

```
[ ]: 0    132.0
1    228.0
2    156.0
3    210.0
4    210.0
5      NaN
6    324.0
7     12.0
8    162.0
9     84.0
Name: age, dtype: float64
```

0.19.1 Dealing with missing values

- in a data set missing values are either? N/A or NaN or 0 or a blank cell.
- Jab kabhi data na ho kisi aik row main kisi b aik parameter ka
 - **Steps:** 1. Koshish karen dobra data collet kar len ya dekh len ager kahin ghalti hy. 2. Missing value wala variable (column) hi nikal den ager data per effect nahe hta ya simple row or data entry remove kar den. 3. Replace the missing values: 1. How? 1. Average value of entire variable or similar data point 2. frequency or MODE replacement 3. Replace based on other functions (Data sampler knows that) 4. ML algorithm can also be used 5. Leave it like that 2. Why? 1. Its better because no data is lost 2. Less accurate

```
[ ]: # where exactly missing values are?
kashti.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

```

```

[ ]: # Use drop.na method
print('Shape Before Removing deck col: ',kashti.shape)
# this will specifically removes deck column
# inplace = true modifies the data frame
kashti.dropna(subset=['deck'], axis=0, inplace=True)
print('Shape Before Removing deck col: ',kashti.shape)

```

```

Shape Before Removing deck col: (891, 15)
Shape Before Removing deck col: (203, 15)

```

```

[ ]: # After removing 'deck' column, see again missing values
kashti.isnull().sum()

```

```

[ ]: survived      0
pclass           0
sex              0
age             19
sibsp           0
parch           0
fare            0
embarked        2
class           0
who             0
adult_male      0
deck            0
embark_town     2
alive           0
alone           0
dtype: int64

```

```

[ ]: # to update the main dataframe
kashti = kashti.dropna() # Removes NAN values from whole dataframe
kashti.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
deck       0
embark_town 0
alive      0
alone      0
dtype: int64
```

```
[ ]: # After dropping all NAN values let see how much data left
print('Shape After Removing all NaN Values: ',kashti.shape)
```

Shape After Removing all NaN Values: (182, 15)

0.19.2 Replacing Missing values with mean of that column

```
[ ]: # Now Using original data again in form of ks1
ks1 = sns.load_dataset('titanic')
ks1.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
```

```
[ ]: # finding an average (mean)
mean_age = ks1['age'].mean()
mean_age
```

```
[ ]: 29.69911764705882
```

```
[ ]: # Replacing NAN values in 'age' column with mean of that column (updating as
      ↪well)
ks1['age'] = ks1['age'].replace(np.nan, mean_age)

# After Replacing age col NaN values with mean of that col
ks1.isnull().sum()
```

```
[ ]: survived      0
     pclass        0
     sex           0
     age           0
     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
```

```
[ ]: ks1.shape
```

```
[ ]: (891, 15)
```

```
[ ]: # Finding datatype of 'deck' column, can we replace missing values with mean or
      ↪not?
ks1.dtypes
```

```
[ ]: survived      int64
     pclass        int64
     sex           object
     age           float64
     sibsp         int64
     parch         int64
     fare          float64
     embarked      object
     class         category
     who           object
     adult_male    bool
     deck          category
     embark_town   object
     alive         object
     alone         bool
```

dtype: object

> As 'deck' is categorical data column, we cannot replace it's NaN values with mean

> So, it's better to remove whole column because it contain too much missing data

```
[ ]: # Removing 'deck' column, Because it's not possible to replace categorical col_
      ↪with mean
print('Shape Before Removing deck col: ',ks1.shape)
ks1 = ks1.drop(['deck'], axis=1)
print('Shape After Removing deck col: ',ks1.shape)
ks1.isnull().sum()
```

Shape Before Removing deck col: (891, 15)

Shape After Removing deck col: (891, 14)

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

0.19.3 Data Formatting

- Data ko aik common standard per lana
- Ensures data is consistant and understandable
 - Easy to gather
 - Easy to workwith
 - Data ek hi unit main ho

```
[ ]: # Know the data type and convert it into the known one
     kashti.dtypes
```

```
[ ]: survived      int64
     pclass        int64
     sex           object
     age           float64
     sibsp         int64
     parch         int64
```



```

fare          float64
embarked      object
class         category
who           object
adult_male    bool
deck          category
embark_town   object
alive         object
alone         bool
dtype: object

```

```

[ ]: # Use this Method to convert datatype / type casting
kashti['survived'] = kashti['survived'].astype('int64')
kashti.dtypes

```

```

[ ]: survived      int64
pclass            int64
sex              object
age              float64
sibsp            int64
parch            int64
fare             float64
embarked         object
class            category
who              object
adult_male       bool
deck             category
embark_town      object
alive            object
alone            bool
dtype: object

```

```

[ ]: # Here We will convert the age into days insted of years
ks1['age_in_days'] = ks1['age'] * 365 # New column added
ks1['age_in_days'] = ks1['age_in_days'].astype('int64')
ks1.head(10)

```

```

[ ]:   survived  pclass   sex      age  sibsp  parch   fare embarked \
0         0        3   male  22.000000    1    0   7.2500         S
1         1        1  female  38.000000    1    0  71.2833         C
2         1        3  female  26.000000    0    0   7.9250         S
3         1        1  female  35.000000    1    0  53.1000         S
4         0        3   male  35.000000    0    0   8.0500         S
5         0        3   male  29.699118    0    0   8.4583         Q
6         0        1   male  54.000000    0    0  51.8625         S
7         0        3   male   2.000000    3    1  21.0750         S
8         1        3  female  27.000000    0    2  11.1333         S

```

```
9          1          2 female 14.000000          1          0 30.0708          C
```

```

class    who  adult_male  embark_town  alive  alone  age_in_days
0  Third    man         True  Southampton    no  False      8030
1  First  woman        False   Cherbourg   yes  False     13870
2  Third  woman        False  Southampton   yes  True      9490
3  First  woman        False  Southampton   yes  False     12775
4  Third    man         True  Southampton   no   True     12775
5  Third    man         True   Queenstown  no   True     10840
6  First    man         True  Southampton   no   True     19710
7  Third  child        False  Southampton   no  False       730
8  Third  woman        False  Southampton   yes  False     9855
9  Second child        False   Cherbourg   yes  False     5110

```

```
[ ]: # Rename the existing Column
ks1.rename(columns= {'age': 'age in days'}, inplace=True)
ks1.head()
```

```
[ ]:   survived  pclass    sex  age in days  sibsp  parch    fare embarked \
0         0         3   male      22.0      1      0    7.2500         S
1         1         1  female      38.0      1      0   71.2833         C
2         1         3  female      26.0      0      0    7.9250         S
3         1         1  female      35.0      1      0   53.1000         S
4         0         3   male      35.0      0      0    8.0500         S

```

```

class    who  adult_male  embark_town  alive  alone  age_in_days
0  Third    man         True  Southampton    no  False      8030
1  First  woman        False   Cherbourg   yes  False     13870
2  Third  woman        False  Southampton   yes  True      9490
3  First  woman        False  Southampton   yes  False     12775
4  Third    man         True  Southampton   no   True     12775

```

0.19.4 Data Normalization

- Uniform the data
- They have same impact
- aik machli samundar main or aik jar main
- Also for Computational reasons

```
[ ]: # Now Using original data again in form of ks2
#ks2 = sns.load_dataset('titanic')
ks1 = ks1.drop(['age in days'], axis=1)
ks1.head()
```

```
[ ]:   survived  pclass    sex  sibsp  parch    fare embarked  class  who \
0         0         3   male      1      0    7.2500         S  Third  man
1         1         1  female      1      0   71.2833         C  First  woman
2         1         3  female      0      0    7.9250         S  Third  woman

```

3	1	1	female	1	0	53.1000	S	First	woman
4	0	3	male	0	0	8.0500	S	Third	man

	adult_male	embark_town	alive	alone	age_in_days
0	True	Southampton	no	False	8030
1	False	Cherbourg	yes	False	13870
2	False	Southampton	yes	True	9490
3	False	Southampton	yes	False	12775
4	True	Southampton	no	True	12775

```
[ ]: ks4 = ks1[['age_in_days', 'fare']]
ks4.head()
```

```
[ ]:   age_in_days    fare
0      8030    7.2500
1     13870   71.2833
2      9490    7.9250
3     12775   53.1000
4     12775    8.0500
```

- The above data is really in wide range, it's hard to compare. So, we need to normalize
- Normalization changes the values to the range of 0-to-1 (now both variable has similar influence on our models)

0.19.5 Method for Normalization

1. Simple Feature Scaling
 1. $x(\text{new}) = x(\text{old}) / x(\text{max})$
2. Min-Max method
3. Z-Score (standard score) -3 -to- +3
4. Log Transformation

```
[ ]: # simple feature scaling
ks4['fare'] = ks4['fare'] / ks4['fare'].max()
ks4['age_in_days'] = ks4['age_in_days'] / ks4['age_in_days'].max()
ks4.head()
```

```
C:\Users\Waleed\AppData\Local\Temp\ipykernel_11444\1927171063.py:2:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
ks4['fare'] = ks4['fare'] / ks4['fare'].max()
C:\Users\Waleed\AppData\Local\Temp\ipykernel_11444\1927171063.py:3:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead
```

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
ks4['age_in_days'] = ks4['age_in_days'] / ks4['age_in_days'].max()
```

```
[ ]:   age_in_days      fare
0      0.2750  0.014151
1      0.4750  0.139136
2      0.3250  0.015469
3      0.4375  0.103644
4      0.4375  0.015713
```

```
[ ]: ks5 = ks1[['age_in_days', 'fare']]
ks5.head()
```

```
[ ]:   age_in_days      fare
0      8030    7.2500
1     13870   71.2833
2      9490    7.9250
3     12775   53.1000
4     12775    8.0500
```

```
[ ]: # min - max method
ks5['fare'] = (ks5['fare'] - ks5['fare'].min()) / (ks5['fare'].max() -
↳ks5['fare'].min())
ks5.head()
```

C:\Users\Waleed\AppData\Local\Temp\ipykernel_11444\1487300651.py:2:

SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using `.loc[row_indexer,col_indexer] = value` instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
ks5['fare'] = (ks5['fare'] - ks5['fare'].min()) / (ks5['fare'].max() -
ks5['fare'].min())
```

```
[ ]:   age_in_days      fare
0      8030  0.014151
1     13870  0.139136
2      9490  0.015469
3     12775  0.103644
4     12775  0.015713
```

```
[ ]: ks6 = ks1[['age_in_days', 'fare']]
ks6.head()
```

```
[ ]:   age_in_days    fare
      0         8030    7.2500
      1        13870   71.2833
      2         9490    7.9250
      3        12775   53.1000
      4        12775    8.0500
```

```
[ ]: # z-score method (standard score)
ks6['fare'] = (ks6['fare'] - ks6['fare'].mean()) / ks6['fare'].std()
ks6.head()
```

C:\Users\Waleed\AppData\Local\Temp\ipykernel_11444\2270294703.py:2:

SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
ks6['fare'] = (ks6['fare'] - ks6['fare'].mean()) / ks6['fare'].std()
```

```
[ ]:   age_in_days    fare
      0         8030 -0.502163
      1        13870  0.786404
      2         9490 -0.488580
      3        12775  0.420494
      4        12775 -0.486064
```

```
[ ]: ks7 = ks1[['age_in_days', 'fare']]
ks7.head()
```

```
[ ]:   age_in_days    fare
      0         8030    7.2500
      1        13870   71.2833
      2         9490    7.9250
      3        12775   53.1000
      4        12775    8.0500
```

```
[ ]: # log transformation method
ks7['fare'] = np.log(ks7['fare'])
ks7.head()
```

c:\Users\Waleed\anaconda3\lib\site-packages\pandas\core\arraylike.py:364:

RuntimeWarning: divide by zero encountered in log

```
result = getattr(ufunc, method)(*inputs, **kwargs)
```

C:\Users\Waleed\AppData\Local\Temp\ipykernel_11444\3833499268.py:2:

SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy

```
ks7['fare'] = np.log(ks7['fare'])
```

```
[ ]:   age_in_days    fare
0      8030    1.981001
1     13870    4.266662
2      9490    2.070022
3     12775    3.972177
4     12775    2.085672
```

0.19.6 Binning

- Grouping of values into smaller number of values (bins)
- Convert numeric into categories (jawan, bachay, booray) etc
- To have better understanding of groups
 - low vs mid vs high price
- bins = number of times the data is being sliced
- labels = the range you are categorizing using labels.

```
[ ]: df = sns.load_dataset('titanic')
      #df.head()
      #print((df['age']).describe())
      bins = np.linspace(min(df['age']), max(df['age']), 4)
      age_groups = ['Bachay', 'Jawan', 'Boorhay']
      df['age'] = pd.cut(df['age'], bins, labels=age_groups, include_lowest=True)
      (df['age']).head()

      # How this will change the names in dataset based on grouping?
```

```
[ ]: 0    Bachay
      1     Jawan
      2    Bachay
      3     Jawan
      4     Jawan
      Name: age, dtype: category
      Categories (3, object): ['Bachay' < 'Jawan' < 'Boorhay']
```

```
[ ]: df.head()
```

```
[ ]:   survived  pclass    sex    age  sibsp  parch    fare embarked  class \
0          0        3   male  Bachay      1      0   7.2500         S   Third
1          1        1  female   Jawan      1      0  71.2833         C   First
2          1        3  female  Bachay      0      0   7.9250         S   Third
3          1        1  female   Jawan      1      0  53.1000         S   First
4          0        3   male   Jawan      0      0   8.0500         S   Third
```

```
who  adult_male  deck  embark_town  alive  alone
```

0	man	True	NaN	Southampton	no	False
1	woman	False	C	Cherbourg	yes	False
2	woman	False	NaN	Southampton	yes	True
3	woman	False	C	Southampton	yes	False
4	man	True	NaN	Southampton	no	True

Converting Categories into dummies - Easy to use for computation - Male Female (0, 1)

```
[ ]: ks1.head()
```

```
[ ]:
survived  pclass    sex  sibsp  parch    fare embarked  class  who \
0         0      3   male     1     0   7.2500         S  Third  man
1         1      1 female     1     0  71.2833         C  First  woman
2         1      3 female     0     0   7.9250         S  Third  woman
3         1      1 female     1     0  53.1000         S  First  woman
4         0      3   male     0     0   8.0500         S  Third  man

adult_male  embark_town  alive  alone  age_in_days
0         True  Southampton   no  False      8030
1        False   Cherbourg  yes  False     13870
2        False  Southampton  yes   True      9490
3        False  Southampton  yes  False     12775
4         True  Southampton   no   True     12775
```

```
[ ]: pd.get_dummies(ks1['sex'])
#ks1.head()
# how to append in dataframe
```

```
[ ]:
0  1
0  0  1
1  1  0
2  1  0
3  1  0
4  0  1
.. .. ..
886 0  1
887 1  0
888 1  0
889 0  1
890 0  1

[891 rows x 2 columns]
```

```
[ ]: #Replace multiple values with multiple new values.
ks1['sex'] = ks1['sex'].replace(['male', 'female'],[1,0])
ks1.head()
```

```

[ ]:  survived  pclass  sex  sibsp  parch      fare embarked  class  who  \
0         0         3    1     1     0   7.2500          S  Third   man
1         1         1    0     1     0  71.2833          C  First  woman
2         1         3    0     0     0   7.9250          S  Third  woman
3         1         1    0     1     0  53.1000          S  First  woman
4         0         3    1     0     0   8.0500          S  Third   man

      adult_male  embark_town  alive  alone  age_in_days
0         True   Southampton    no  False      8030
1        False    Cherbourg   yes  False     13870
2        False   Southampton   yes   True      9490
3        False   Southampton   yes  False     12775
4         True   Southampton    no   True     12775

```