

# chilla with Baba Aammar

\*\*Muhammad Fahad\*\*

Bachelor's in Computer Science

## Table of Content

### 1. Basic Python

- 1.1 Operators
- 1.2 Strings
- 1.3 Variables
- 1.4 Input\_Variables
- 1.5 Type\_Conversion
- 1.6 Conditional\_Statements
- 1.7 Functions
- 1.8 Loops
  - 1.8.1 while-Loop
  - 1.8.2 for-loop
- 1.9 Arrays
- 1.10 Libraries
- 1.11 Errors trouble\_shooting

### 1. Data Structures

- 2.1 Indexing
- 2.2 Strings

### 1. Basic Data structures in Python

- 3.1 Tuples
- 3.2 List
- 3.3 Dictionary

- 3.4 Sets

## 1. Numpy

- 4.1 1D-array
- 4.2 2D-array
- 4.3 3D-array
- 4.4 More Practice

## 1. Pandas

- 5.1 PANDAS Hand's ON
  - 5.1.1 Sorting
- 5.2 Case Study with PANDAS
- 5.3 (Land) FAO Dataset

## 1. Statistics

- 6.1 (EDA).Exploratory Data Analysis
  - 6.1.1 01\_Undestand the Data
  - 6.1.2 02\_Cleaning & Filtering the Data
  - 6.1.3 03\_Find missing values
  - 6.1.4 04\_Remove missing values
  - 6.1.5 05\_Remove Outliers
- 6.2 Plots
- 6.3 Relationship between Data
- 6.4 Data Preprocessing or Data Wrangling
  - 6.4.1 Simple Operation
  - 6.4.2 Find missing values
  - 6.4.3 Find shape of Dataset
  - 6.4.4 Remove missing values from whole Dataset
  - 6.4.5 Simple Operation
- 6.5 Data Cleaning
- 6.6 Data Normalization
  - 6.6.1 Method of Normalize Data
    - 6.6.1.1 Simple Feature
    - 6.6.1.2 Min-Max Method

- 6.6.1.3 Z-score. (Standard Score)
- 6.6.1.4 log transformation
- 6.7 [Binning](#)

## 1. [Machine Learning](#)

- 7.1 [Simple\\_Linear\\_Regression](#)
- 7.2 [Multiple\\_Linear\\_Regression](#)
- 7.3 [Decision\\_Tree\\_Classifier](#)
- 7.4 [Random-Forest\\_Classifier](#)
- 7.5 [K-Nearest\\_Neighbors](#)
- 7.6 [Polynomial Regression](#)
- 7.7 [Naive Bayes](#)
- 7.8 [Support\\_Vector\\_Machine](#)
- 7.9 [Clustering](#)

# DataScience with Python | Crash Course

## (1). PYTHON Basics

### 1st Program with PY !

In [1]:

```
print("Fahad")
print("COMSATS University Islamabad (Sahiwal Campus)")
print(2+3)
```

```
Fahad
COMSATS University Islamabad (Sahiwal Campus)
5
```

### operators.py

In [2]:

```
print("Add:", 42+55)
print("Mul:", 2*2)
print("Mod:", 9%2)
print("Div with Float:", 10/5)
```

```
print("Div without Float:", 10//5)
print("Sub:", 4-2)
print("Power:", 2**3)

print("PEMDAS Rule: ", 8+5-2*6-8/8+6/2*4)
```

```
Add: 97
Mul: 4
Mod: 1
Div with Float: 2.0
Div without Float: 2
Sub: 2
Power: 8
PEMDAS Rule: 12.0
```

## strings.py

In [3]:

```
print("Muhammad Fahad")
print('PAKISTAN !')
print('SP19-BCS-042')
```

```
Muhammad Fahad
PAKISTAN !
SP19-BCS-042
```

## variables.py

In [4]:

```
# (1)
f=42
print(f)
print(type(f))

# (2)
x="Myself Fahad"
print(x)
print(type(x))

# (3)
y=5.5
print(y)
print(type(y))
```

```
# (4)
fruit = "Apples, Bananas"
print(fruit)

# (5)
# vegetables = "Carrot, Potato"
# del vegetables
# print(vegetables)
```

```
42
<class 'int'>
Myself Fahad
<class 'str'>
5.5
<class 'float'>
Apples, Bananas
```

## input\_variables.py

In [5]:

```
# (1)
# age=input("What's your Age: ")
# print(age)

# (2)
# fruit_basket=input("Which fruit is your favorite: ")
# print("I like", fruit_basket)

# (3)
name="Faadie!"
welcome="Hy"

print(welcome, name)
print("ALi")

# (4)
# name=input("What's your Good Name:")
# age=input("What's your Age:")
# greetings= "Welcome! "

# print(greetings, name, ", Younger Boy", age)
```

Hy Faadie!  
ALi

## conditional\_logics\_operators.py

In [6]:

```
school_age=5

# (1)
# child_age=4
# print(child_age == school_age)

# (2)
child_age=input("What's your Child Age:")
print(type(child_age))
child_age=int(child_age)
print(type(child_age))
print(child_age >= school_age)
```

```
What's your Child Age:23
<class 'str'>
<class 'int'>
True
```

## type\_conversion.py

In [7]:

```
# (1)
age=22
print("Age: ", age, type(str(age)))

# (2)
reg="42"
print(reg, type(reg))

# (3)
reg=float(reg)
print("reg", reg, type(reg))
```

```
Age: 22 <class 'str'>
42 <class 'str'>
reg 42.0 <class 'float'>
```

## (if\_else elif)\_condition\_statements.py

In [8]:

```

school_age=5
college_age=16
uni_age=18

child_age=input("What's your Child Age:")
child_age=int(child_age)

# (1)
if child_age >= school_age and child_age < college_age:
    print("You Join School !")
# (2)
elif child_age >= college_age and child_age < uni_age:
    print("You Join College !")
elif child_age >= uni_age:
    print("You Join University OR Doing Work well in Market!")
# (3)
else:
    print("Please care of your Baby !!!")

```

What's your Child Age:23  
 You Join University OR Doing Work well in Market!

## functions.py

In [9]:

```

# # (1)
# def home():
#     print("House: 1419")
# home()

# # (2)
# def shop():
#     text="Okanwala Road CCW"
#     print(text)
#     print(text)
#     print(text)
# shop()

# # (3)
# def uni(place):
#     print("COMSATS University!", place)
# uni("Sahiwal")

```

```
# print()
# print()
# print()

# (4)
# print("MADE SCHOOL CALCULATOR FOR KIDS !")
# school_age=5
# college_age=16
# uni_age=18

# def school_age_calculator(child_age):
#     if child_age >= school_age and child_age < college_age:
#         print("You Join School !")
#     elif child_age >= college_age and child_age < uni_age:
#         print("You Join College !")
#     elif child_age >= uni_age:
#         print("You Join University OR Doing Work well in Market!")
#     else:
#         print("Please care of your Baby !!!")

# school_age_calculator(16)

# # (5)
# print("MADE SCHOOL CALCULATOR FOR KIDS !")
# school_age=5
# college_age=16
# uni_age=18

# child_age = input("What's your Age: ")
# child_age = int(child_age)
# def school_age_calculator(text, child_age):
#     if child_age >= school_age and child_age < college_age:
#         print("You Join School !")
#     elif child_age >= college_age and child_age < uni_age:
#         print("You Join College !")
#     elif child_age >= uni_age:
#         print("You Join University OR Doing Work well in Market!")
#     else:
#         print("Please care of your Baby !!!")

# school_age_calculator("Your age is: ", child_age)
```

```
# # (6)
# print("Age-Prediction in Future !")

# def future_Age():
#     new_age = age+10
#     print("Future Prediction: ", new_age)
# future_Age(22)

# (7)
print("Age-Prediction in Future !")

def future_Age(age):
    new_age = age+3
    return new_age

future_Age(22)
```

Out[9]:  
Age-Prediction in Future !  
25

## loops.py

### While-Loop

In [10]:

```
# (1)
x=1
while(x<=5):
    print("No:", x)
    x=x+1

# (2)
# y=10
# while(y>0):
#     print("No:", y)
#     y=y-1
```

No: 1  
No: 2  
No: 3

No: 4

No: 5

## For-Loop

In [11]:

```
# # (1)
# for x in range(1, 5):
#     print(x)

# (2)
for y in range(10, 16):
    print(y)
```

```
10
11
12
13
14
15
```

## arrays.py

In [12]:

```
# (1)
# array !
brothers = ["Tayyab", "Adil", "Tahar", "Fahad"]
for bros in brothers:
    print("Brothers are:", bros)

# (2)
# array !
# days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
# for d in days:
#     print("Days are:", d)

# # (3)
# # array !
# days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]
# for d in days:
#     if(d=="Fri"): continue # Skip !
#     if(d=="Wed"): break # Loop Stops !
```

```
#     print("Days are:", d)
```

Brothers are: Tayyab  
Brothers are: Adil  
Brothers are: Tahar  
Brothers are: Fahad

## libraries.py

In [13]:

```
# Important Libraries.(numpy, pandas)

# (1)
# import math
# print("The Value of 'pi' is:", math.pi)
# print("The Value of 'celsius' is:", math.ceil)

# (2)
import statistics
x=[150, 200, 250, 100]
print("Mean:", statistics.mean(x))
print("Median:", statistics.median(x))
```

Mean: 175  
Median: 175.0

## trouble\_shoot\_errors.py

In [14]:

```
# Syntax Error !
# Run-time Error !
# Logical Error !
```

# (2). Data Structures

## - Indexing

```
In [15]: green = "Pakistan Zindabad"  
green
```

```
Out[15]: 'Pakistan Zindabad'
```

```
In [16]: green[0]
```

```
Out[16]: 'P'
```

```
In [17]: green[9:17]
```

```
Out[17]: 'Zindabad'
```

```
In [18]: green[5]
```

```
Out[18]: 't'
```

```
In [19]: green[-1]
```

```
Out[19]: 'd'
```

```
In [20]: len(green)
```

```
Out[20]: 17
```

## - Strings

```
In [21]: food = "Biryani"  
print("Food: "+food)
```

```
Food: Biryani
```

```
In [22]: food.find("i")
```

Out[22]: 1

In [23]: len(food)

Out[23]: 7

In [24]: food.upper()

Out[24]: 'BIRYANI'

In [25]: food.lower()

Out[25]: 'biryani'

In [26]: food.replace("B", "Sh")

Out[26]: 'Shiryani'

In [27]: food.casefold()

Out[27]: 'biryani'

In [28]: name = "muhammad Fahad"  
name

Out[28]: 'muhammad Fahad'

In [29]: name.count("a")

Out[29]: 4

In [30]: name.count("m")

Out[30]: 3

```
In [31]: name.find("m")
```

```
Out[31]: 0
```

```
In [32]: name.find("d")
```

```
Out[32]: 7
```

## - Finding an index number in string

```
In [33]: name.find("f")
```

```
Out[33]: -1
```

```
In [34]: taste = "I love Chicken Burger, Sandwitch, Pizza, Fries and Rice !"  
taste
```

```
Out[34]: 'I love Chicken Burger, Sandwitch, Pizza, Fries and Rice !'
```

```
In [35]: taste.split(",")
```

```
Out[35]: ['I love Chicken Burger', ' Sandwitch', ' Pizza', ' Fries and Rice !']
```

```
In [36]: taste.split("a")
```

```
Out[36]: ['I love Chickn Burger, S', 'ndwitch, Pizz', ', Fries ', 'nd Rice !']
```

## (3). Basic Data structures in Python

=> **Tuple** .( Un-Mutable, Not changeable/modifiable )

=> **List** .[ Muteable, changeable/modifiable ]

=> **Dictionary** .{ Muteable, changeable/modifiable }

=> **Set** .{No-Duplication, Unordered Indexed, Boolean-type not Allowed}

## => 1.Tuple... ()

In [37]:

```
set_tuple = (42, "COMSATS", 42.5, 1)
print("Tuple():", set_tuple)
type(set_tuple)
```

Tuple(): (42, 'COMSATS', 42.5, 1)

Out[37]:

tuple

In [38]:

```
set_tuple[1]
set_tuple[0:2]
```

Out[38]:

(42, 'COMSATS')

In [39]:

```
len(set_tuple)
```

Out[39]:

4

In [40]:

```
set_tuple1 = (55, "LUMS", 45.5, False)
print("Tuple():", set_tuple)
type(set_tuple)
```

Tuple(): (42, 'COMSATS', 42.5, 1)

Out[40]:

tuple

In [41]:

```
set_tuple + set_tuple1
```

Out[41]:

(42, 'COMSATS', 42.5, 1, 55, 'LUMS', 45.5, False)

In [42]:

```
set_tuple*2 + set_tuple1
```

```
Out[42]: (42, 'COMSATS', 42.5, 1, 42, 'COMSATS', 42.5, 1, 55, 'LUMS', 45.5, False)
```

```
In [43]: tp = (42, 55, 47, 50, 12, 10 ,18)
```

```
In [44]: min(tp)
```

```
Out[44]: 10
```

```
In [45]: max(tp)
```

```
Out[45]: 55
```

```
In [46]: tp*2
```

```
Out[46]: (42, 55, 47, 50, 12, 10, 18, 42, 55, 47, 50, 12, 10, 18)
```

```
In [47]: tp*3
```

```
Out[47]: (42,  
55,  
47,  
50,  
12,  
10,  
18,  
42,  
55,  
47,  
50,  
12,  
10,  
18,  
42,  
55,  
47,  
50,  
12,
```

```
10,  
18)
```

## => 2.List... [ ]

```
In [48]:  
list1 = [1, "Fahad", 42, True]  
list1
```

```
Out[48]: [1, 'Fahad', 42, True]
```

```
In [49]: type(list1)
```

```
Out[49]: list
```

```
In [50]: list1[1:3]
```

```
Out[50]: ['Fahad', 42]
```

```
In [51]:  
list2 = [55, 42, 12, 18, 47, 48, 26, 31, 35, 42, 33]  
list2
```

```
Out[51]: [55, 42, 12, 18, 47, 48, 26, 31, 35, 42, 33]
```

```
In [52]: list1 + list2
```

```
Out[52]: [1, 'Fahad', 42, True, 55, 42, 12, 18, 47, 48, 26, 31, 35, 42, 33]
```

```
In [53]: list1.reverse()
```

```
In [54]: list1
```

```
Out[54]: [True, 42, 'Fahad', 1]
```

```
In [55]: list2.sort()
```

```
In [56]: list2
```

```
Out[56]: [12, 18, 26, 31, 33, 35, 42, 42, 47, 48, 55]
```

```
In [57]: list2.count(42)
```

```
Out[57]: 2
```

```
In [58]: list1.append("reg")
```

```
In [59]: list1
```

```
Out[59]: [True, 42, 'Fahad', 1, 'reg']
```

```
In [60]: len(list2)
```

```
Out[60]: 11
```

## => 3.Dictionary... { }

```
In [61]: dict = {"pizza": 400, "burger": 200, "Fries": 100, "Shawarma": 130, "Paratha Roll": 180}
dict1 = {"eng": 100, "urdu": 70, "chem": 150, "phy": 170, "mth": 200}
```

```
In [62]: print("Food:", dict)
print("Books:", dict1)
```

```
Food: {'pizza': 400, 'burger': 200, 'Fries': 100, 'Shawarma': 130, 'Paratha Roll': 180}
Books: {'eng': 100, 'urdu': 70, 'chem': 150, 'phy': 170, 'mth': 200}
```

```
In [63]: type(dict1)
```

```
Out[63]: dict
```

```
In [64]: keys_dict = dict.keys()
values_dict = dict.values()

print("Keys of Food:", keys_dict)
print("Values of Food:", values_dict)
```

Keys of Food: dict\_keys(['pizza', 'burger', 'Fries', 'Shawarma', 'Paratha Roll'])  
 Values of Food: dict\_values([400, 200, 100, 130, 180])

```
In [65]: dict1["bio"] = 250
dict1
```

Out[65]: {'eng': 100, 'urdu': 70, 'chem': 150, 'phy': 170, 'mth': 200, 'bio': 250}

```
In [66]: dict1["bio"] = 300
dict1
```

Out[66]: {'eng': 100, 'urdu': 70, 'chem': 150, 'phy': 170, 'mth': 200, 'bio': 300}

```
In [67]: # merge the 2 Dictionaries !
dict.update(dict1)
dict
```

Out[67]: {'pizza': 400,
 'burger': 200,
 'Fries': 100,
 'Shawarma': 130,
 'Paratha Roll': 180,
 'eng': 100,
 'urdu': 70,
 'chem': 150,
 'phy': 170,
 'mth': 200,
 'bio': 300}

## => 4.Sets... ()

```
In [68]: data = {1, 1419, "Fahad", "COMSATS", True, "CCW"}
data
```

```
Out[68]: {1, 1419, 'CCW', 'COMSATS', 'Fahad'}
```

```
In [69]: data.add(55)  
data
```

```
Out[69]: {1, 1419, 55, 'CCW', 'COMSATS', 'Fahad'}
```

```
In [70]: data.remove(1419)  
data
```

```
Out[70]: {1, 55, 'CCW', 'COMSATS', 'Fahad'}
```

```
In [71]: data
```

```
Out[71]: {1, 55, 'CCW', 'COMSATS', 'Fahad'}
```

## (4). Numpy Practice

```
In [72]: import numpy as np
```

### 1D-array. (Vector Array)

#### Creating an array using numpy

```
In [73]: food = np.array(["pizza", "burger", "roll"])  
food
```

```
Out[73]: array(['pizza', 'burger', 'roll'], dtype='<U6')
```

```
In [74]: food.ndim
```

```
Out[74]: 1
```

```
In [75]: type(food)
```

```
Out[75]: numpy.ndarray
```

```
In [76]: len(food)
```

```
Out[76]: 3
```

```
In [77]: food[1]
```

```
Out[77]: 'burger'
```

```
In [78]: food[1:]
```

```
Out[78]: array(['burger', 'roll'], dtype='<U6')
```

---

```
In [79]: number = np.array([42, 55, 47, 50, 26])
number
```

```
Out[79]: array([42, 55, 47, 50, 26])
```

```
In [80]: type(number)
```

```
Out[80]: numpy.ndarray
```

```
In [81]: len(number)
```

```
Out[81]: 5
```

```
In [82]: number[3]
```

```
Out[82]: 50
```

```
In [83]: number[2:]
```

```
Out[83]: array([47, 50, 26])
```

```
In [84]: number.mean() # mean method !
```

```
Out[84]: 44.0
```

```
In [85]: np.zeros(4) # zeroes method !
```

```
Out[85]: array([0., 0., 0., 0.])
```

```
In [86]: np.ones(6) # ones method !
```

```
Out[86]: array([1., 1., 1., 1., 1., 1.])
```

```
In [87]: np.empty(3)
```

```
Out[87]: array([0., 0., 0.])
```

### ***arange method !***

```
In [88]: np.arange(8)
```

```
Out[88]: array([0, 1, 2, 3, 4, 5, 6, 7])
```

```
In [89]: np.arange(1,10)
```

```
Out[89]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
In [90]: np.arange(5,55,5) # table of (5)
```

```
Out[90]: array([ 5, 10, 15, 20, 25, 30, 35, 40, 45, 50])
```

```
In [91]: np.linspace(1,20,10)
```

```
Out[91]: array([ 1.          ,  3.11111111,  5.22222222,  7.33333333,  9.44444444,
   11.55555556, 13.66666667, 15.77777778, 17.88888889, 20.        ])
```

```
In [92]: np.sort(number)
```

```
Out[92]: array([26, 42, 47, 50, 55])
```

## Array-Functions

```
In [93]: a = np.array([5, 6, 9, 50, 1, 30, 24, 14, 2, 42, 26]) # integer array
a
```

```
Out[93]: array([ 5,  6,  9, 50,  1, 30, 24, 14,  2, 42, 26])
```

```
In [94]: a.ndim
```

```
Out[94]: 1
```

```
In [95]: b = np.array([14.1, 13.5, 35.1, 2.71, 3.05, 26.4, 42.7, 18.3, 40.4, 10.2]) # floating-point array
b
```

```
Out[95]: array([14.1 , 13.5 , 35.1 ,  2.71,  3.05, 26.4 , 42.7 , 18.3 , 40.4 ,
   10.2 ])
```

```
In [96]: a.sort()
a
```

```
Out[96]: array([ 1,  2,  5,  6,  9, 14, 24, 26, 30, 42, 50])
```

```
In [97]: b.sort()
b
```

```
Out[97]: array([ 2.71,  3.05, 10.2 , 13.5 , 14.1 , 18.3 , 26.4 , 35.1 , 40.4 ,
   42.7 ])
```

```
In [98]: c = np.concatenate([a,b])
c
```

```
Out[98]: array([ 1. ,  2. ,  5. ,  6. ,  9. , 14. , 24. , 26. , 30. ,
   42. , 50. , 2.71, 3.05, 10.2 , 13.5 , 14.1 , 18.3 , 26.4 ,
   35.1 , 40.4 , 42.7 ])
```

```
In [99]: np.sort(c)
```

```
Out[99]: array([ 1. ,  2. ,  2.71,  3.05,  5. ,  6. ,  9. , 10.2 , 13.5 ,
   14. , 14.1 , 18.3 , 24. , 26. , 26.4 , 30. , 35.1 , 40.4 ,
   42. , 42.7 , 50. ])
```

```
In [100... c.ndim
```

```
Out[100... 1
```

## 2D-array. (Matrix Array)

```
In [101... a = np.array([[1,2,2],[3,5,3],[11,22,33]])
a
```

```
Out[101... array([[ 1,  2,  2],
   [ 3,  5,  3],
   [11, 22, 33]])
```

```
In [102... b = np.array([[4,6,4],[5,2,3],[2,3,5]])
b
```

```
Out[102... array([[ 4,  6,  4],
   [ 5,  2,  3],
   [ 2,  3,  5]])
```

```
In [103... c = np.concatenate([a,b], axis=0) # Length would be same !
```

c

```
Out[103... array([[ 1,  2,  2],
       [ 3,  5,  3],
       [11, 22, 33],
       [ 4,  6,  4],
       [ 5,  2,  3],
       [ 2,  3,  5]])
```

```
In [104... d = np.concatenate([a,b], axis=1) # Length would be same !
d
```

```
Out[104... array([[ 1,  2,  2,  4,  6,  4],
       [ 3,  5,  3,  5,  2,  3],
       [11, 22, 33,  2,  3,  5]])
```

```
In [105... d.ndim
```

```
Out[105... 2
```

```
In [106... np.reshape(d, newshape=(1,18), order='C')
```

```
Out[106... array([[ 1,  2,  2,  4,  6,  4,  3,  5,  3,  5,  2,  3, 11, 22, 33,  2,
       3,  5]])
```

---

## 3D-array. (Tensor Array)

```
In [107... x = np.array([[[1,2],[3,4]],
       [[5,6],[7,8]],
       [[9,1],[4,8]]])
x
```

```
Out[107... array([[[1, 2],
       [3, 4]],

       [[5, 6],
       [7, 8]],
```

```
[[9, 1],  
 [4, 8]])
```

In [108...]: `x.ndim`

Out[108...]: 3

In [109...]: `type(x)`

Out[109...]: `numpy.ndarray`

In [110...]: `np.size(x)`

Out[110...]: 12

In [111...]: `x`

Out[111...]: `array([[1, 2],  
 [3, 4],  
  
 [[5, 6],  
 [7, 8]],  
  
 [[9, 1],  
 [4, 8]])`

In [112...]: `x.shape`

Out[112...]: (3, 2, 2)

In [113...]: `y = np.array([[9,8],[7,6]],  
 [[5,4],[3,2]],  
 [[1,2],[4,3]]))`  
 y

Out[113...]: `array([[9, 8],  
 [7, 6]],`

```
[[5, 4],  
 [3, 2]],
```

```
[[1, 2],  
 [4, 3]])
```

```
In [114... y.ndim
```

```
Out[114... 3
```

```
In [115... type(y)
```

```
Out[115... numpy.ndarray
```

```
In [116... np.size(y)
```

```
Out[116... 12
```

```
In [117... y
```

```
Out[117... array([[9, 8],  
 [7, 6]],
```

```
[[5, 4],  
 [3, 2]],
```

```
[[1, 2],  
 [4, 3]])
```

```
In [118... y.shape
```

```
Out[118... (3, 2, 2)
```

## ***orange method !***

```
In [119... a = np.arange(27) # orange [27]  
a
```

```
Out[119... array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
   17, 18, 19, 20, 21, 22, 23, 24, 25, 26])
```

```
In [120... t = a.reshape(3,3,3) # reshape = [3*3*3 = 27]
t
```

```
Out[120... array([[[ 0,  1,  2],
   [ 3,  4,  5],
   [ 6,  7,  8]],

   [[ 9, 10, 11],
   [12, 13, 14],
   [15, 16, 17]],

   [[18, 19, 20],
   [21, 22, 23],
   [24, 25, 26]]])
```

---

```
In [121... t = np.arange(27).reshape(3,3,3)
t
```

```
Out[121... array([[[ 0,  1,  2],
   [ 3,  4,  5],
   [ 6,  7,  8]],

   [[ 9, 10, 11],
   [12, 13, 14],
   [15, 16, 17]],

   [[18, 19, 20],
   [21, 22, 23],
   [24, 25, 26]]])
```

---

### convert 1D-array to 2D-array

```
In [122... u = np.array([5,6,8,9,4,3,2])
u
```

```
Out[122... array([5, 6, 8, 9, 4, 3, 2])
```

```
In [123... u[2:5]
```

```
Out[123... array([8, 9, 4])
```

```
In [124... u[:6]
```

```
Out[124... array([5, 6, 8, 9, 4, 3])
```

```
In [125... u[1:]
```

```
Out[125... array([6, 8, 9, 4, 3, 2])
```

```
In [126... u.sum()
```

```
Out[126... 37
```

```
In [127... u.mean()
```

```
Out[127... 5.285714285714286
```

```
In [128... u*2
```

```
Out[128... array([10, 12, 16, 18, 8, 6, 4])
```

```
In [129... u+1
```

```
Out[129... array([ 6,  7,  9, 10,  5,  4,  3])
```

```
In [130... u.shape
```

```
Out[130... (7,)
```

```
In [131... u.ndim
```

Out[131... 1

In [132...

```
# row-wise (2D)
o = u[np.newaxis, :]
o
```

Out[132...

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

In [133...

```
# columns-wise (2D)
o = u[:, np.newaxis]
o
```

Out[133...

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

In [134...

```
o.shape
```

Out[134...

```
(7, 1)
```

In [135...

```
o.ndim
```

Out[135...

```
2
```

## More Practice

In [136...

```
import numpy as np
```

## 1D-array

In [137...

```
p = np.array([42, 55, 50]) # 1D-array / Vector
p
```

```
Out[137... array([42, 55, 50])
```

```
In [138... aa = np.array([5, 6, 1, 8])
bb = np.array([16, 19, 50, 100, 11, 3])
cc = np.concatenate((aa, bb))
cc
```

```
Out[138... array([ 5,  6,  1,  8, 16, 19, 50, 100, 11,  3])
```

```
In [139... np.sort(cc)
```

```
Out[139... array([ 1,  3,  5,  6,  8, 11, 16, 19, 50, 100])
```

```
In [140... np.sort(p)
```

```
Out[140... array([42, 50, 55])
```

```
In [141... type(p)
```

```
Out[141... numpy.ndarray
```

```
In [142... len(p)
```

```
Out[142... 3
```

```
In [143... p[0]
```

```
Out[143... 42
```

```
In [144... res = np.array([1, 2, 3, 4, 5, 6]) # array
res
```

```
Out[144... array([1, 2, 3, 4, 5, 6])
```

```
In [145...]  
a = np.zeros(2) # for zeroes 0's  
a
```

```
Out[145...]  
array([0., 0.])
```

```
In [146...]  
b = np.ones(3) # for ones 1's  
b
```

```
Out[146...]  
array([1., 1., 1.])
```

```
In [147...]  
c = np.empty(5) # empty  
c
```

```
Out[147...]  
array([0.00e+000, 0.00e+000, 0.00e+000, 9.92e-321, 0.00e+000])
```

```
In [148...]  
d = np.arange(9) # range to (9)  
d
```

```
Out[148...]  
array([0, 1, 2, 3, 4, 5, 6, 7, 8])
```

```
In [149...]  
e = np.arange(1, 56) # range b/w (1 to 56)  
e
```

```
Out[149...]  
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,  
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,  
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,  
       52, 53, 54, 55])
```

```
In [150...]  
f = np.arange(0, 30, 3) # range with difference 3's b/w (0 to 30)  
f
```

```
Out[150...]  
array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27])
```

```
In [151...]  
g = np.linspace(2, 20, num=10) # linearly spaced array  
g
```

```
Out[151...]  
array([ 2.,  4.,  6.,  8., 10., 12., 14., 16., 18., 20.])
```

```
In [152...  
h = np.ones(5, dtype=np.int8) # choose specific- dataType  
h
```

```
Out[152... array([1, 1, 1, 1, 1], dtype=int8)
```

```
In [153...  
i = np.ones(4, dtype=np.float16) # choose specific- dataType  
i
```

```
Out[153... array([1., 1., 1., 1.], dtype=float16)
```

## 2D-array

```
In [154...  
r = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]]) # 2d-array  
r
```

```
Out[154... array([[1, 2, 3],  
[4, 5, 6],  
[7, 8, 9]])
```

```
In [155...  
type(r)
```

```
Out[155... numpy.ndarray
```

```
In [156...  
len(r)
```

```
Out[156... 3
```

```
In [157...  
r[2]
```

```
Out[157... array([7, 8, 9])
```

```
In [158...  
x = np.zeros([2,3])  
x
```

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

In [158... Out[158... [0., 0., 0.]])

In [159... xx = np.ones([3,2])  
xx

Out[159... array([[1., 1.],  
[1., 1.],  
[1., 1.]])

In [160... y = np.empty([3,3])  
y

Out[160... array([[1.06137775e-312, 0.00000000e+000, 1.06395697e+224],  
[6.58640545e-066, 3.24865263e-057, 5.18589698e+170],  
[1.17448468e+165, 3.44104181e-085, 6.27207943e-310]])

In [161... k = np.array([5, 6 ,1 ,8])  
j = np.array([16, 19, 50, 100, 11, 3])  
pp = np.concatenate([k, j])  
pp

Out[161... array([ 5, 6, 1, 8, 16, 19, 50, 100, 11, 3])

## 3D-array

In [162... dd = np.arange(45).reshape(3,5,3)  
dd

Out[162... array([[[ 0, 1, 2],  
[ 3, 4, 5],  
[ 6, 7, 8],  
[ 9, 10, 11],  
[12, 13, 14]],  
  
[[15, 16, 17],  
[18, 19, 20],  
[21, 22, 23],  
[24, 25, 26],  
[27, 28, 29]],  
  
[[30, 31, 32],

```
[33, 34, 35],  
[36, 37, 38],  
[39, 40, 41],  
[42, 43, 44]]))
```

In [163...]  
ddd = np.arange(24).reshape(4,2,3)  
ddd

Out[163...]  
array([[[ 0, 1, 2],  
 [ 3, 4, 5]],  
  
 [[ 6, 7, 8],  
 [ 9, 10, 11]],  
  
 [[12, 13, 14],  
 [15, 16, 17]],  
  
 [[18, 19, 20],  
 [21, 22, 23]]])

In [164...]  
t = np.arange(30).reshape(5,2,3)  
t

Out[164...]  
array([[[ 0, 1, 2],  
 [ 3, 4, 5]],  
  
 [[ 6, 7, 8],  
 [ 9, 10, 11]],  
  
 [[12, 13, 14],  
 [15, 16, 17]],  
  
 [[18, 19, 20],  
 [21, 22, 23]],  
  
 [[24, 25, 26],  
 [27, 28, 29]]])

---

---

## (5). Pandas Practice

In [165...]

```
# Libraries !
import pandas as pd
import numpy as np
import stats
import matplotlib.pyplot as plt
import seaborn as sns
```

In [166...]

```
a = np.array([1,5,10,10,15,20])
a
```

Out[166...]

```
array([ 1,  5, 10, 10, 15, 20])
```

In [167...]

```
np.mean(a)
```

Out[167...]

```
10.166666666666666
```

In [168...]

```
np.median(a)
```

Out[168...]

```
10.0
```

In [169...]

```
stats.mode(a)
```

Out[169...]

```
10
```

In [170...]

```
flowers = sns.load_dataset("iris")
flowers.head()
```

Out[170...]

	sepal_length	sepal_width	petal_length	petal_width	species
<b>0</b>	5.1	3.5	1.4	0.2	setosa
<b>1</b>	4.9	3.0	1.4	0.2	setosa
<b>2</b>	4.7	3.2	1.3	0.2	setosa
<b>3</b>	4.6	3.1	1.5	0.2	setosa
<b>4</b>	5.0	3.6	1.4	0.2	setosa

In [171...]

```
flowers.to_csv("iris.csv")
```

In [172...]

```
flowers = pd.read_csv("iris.csv")
flowers.head()
```

Out[172...]

	Unnamed: 0	sepal_length	sepal_width	petal_length	petal_width	species
<b>0</b>	0	5.1	3.5	1.4	0.2	setosa
<b>1</b>	1	4.9	3.0	1.4	0.2	setosa
<b>2</b>	2	4.7	3.2	1.3	0.2	setosa
<b>3</b>	3	4.6	3.1	1.5	0.2	setosa
<b>4</b>	4	5.0	3.6	1.4	0.2	setosa

In [173...]

```
flowers.describe()
```

Out[173...]

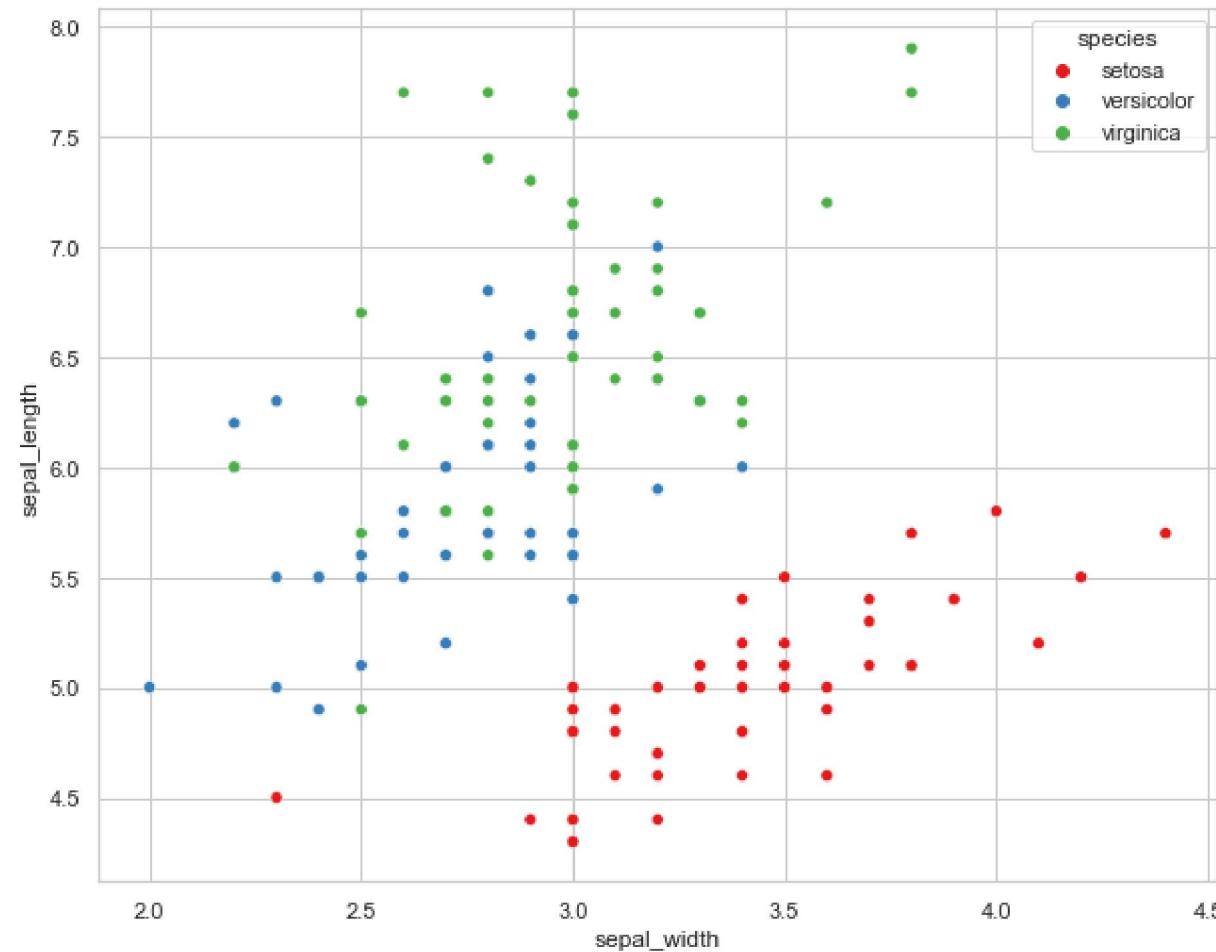
	Unnamed: 0	sepal_length	sepal_width	petal_length	petal_width
<b>count</b>	150.000000	150.000000	150.000000	150.000000	150.000000
<b>mean</b>	74.500000	5.843333	3.057333	3.758000	1.199333
<b>std</b>	43.445368	0.828066	0.435866	1.765298	0.762238
<b>min</b>	0.000000	4.300000	2.000000	1.000000	0.100000
<b>25%</b>	37.250000	5.100000	2.800000	1.600000	0.300000
<b>50%</b>	74.500000	5.800000	3.000000	4.350000	1.300000
<b>75%</b>	111.750000	6.400000	3.300000	5.100000	1.800000
<b>max</b>	149.000000	7.900000	4.400000	6.900000	2.500000

In [174...]

```
plt.figure(figsize=(10,8))

sns.set_theme(style="whitegrid", color_codes=True)
sns.scatterplot(x="sepal_width",
                 y="sepal_length",
```

```
hue="species",
data=flowers,
palette="Set1")
plt.show()
```



## PANDAS Hand's ON

In [175...]

```
# import Libraries !
import pandas as pd
import numpy as np
```

In [176...]

```
a = pd.array([1, 2, np.nan, 5, 6, 7, 2, 4, 9])
```

```
a
```

```
Out[176... <IntegerArray>
[1, 2, <NA>, 5, 6, 7, 2, 4, 9]
Length: 9, dtype: Int64
```

```
In [177... a = pd.Series(a) # Columns
a
```

```
Out[177... 0      1
1      2
2    <NA>
3      5
4      6
5      7
6      2
7      4
8      9
dtype: Int64
```

---

```
In [178... b = pd.Series([1, 2, np.nan, 5, 6, 7, 2, 4, 9]) # Columns
b
```

```
Out[178... 0    1.0
1    2.0
2    NaN
3    5.0
4    6.0
5    7.0
6    2.0
7    4.0
8    9.0
dtype: float64
```

```
In [179... b = pd.array(b)
b
```

```
Out[179... <PandasArray>
[1.0, 2.0, nan, 5.0, 6.0, 7.0, 2.0, 4.0, 9.0]
Length: 9, dtype: float64
```

```
In [180... dates = pd.date_range("20220101", periods=10) # dates
dates
```

```
Out[180... 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 [181... # df stands data-frame contains both (Rows & Columns) !
df = pd.DataFrame(np.random.randn(10,5),
                  index=dates,
                  columns=list("FAHAD"),
                  dtype="float64")

df
```

	F	A	H	A	D
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716
<b>2022-01-03</b>	2.223851	0.763401	-1.298154	1.100876	1.277980
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	0.006742	-0.001046
<b>2022-01-05</b>	-0.194993	-0.323621	-0.400696	1.642857	-0.179146
<b>2022-01-06</b>	-1.176169	-0.527270	-0.095128	1.321913	-0.707781
<b>2022-01-07</b>	0.102521	-0.339865	-1.176499	1.476894	0.665398
<b>2022-01-08</b>	-0.546442	1.204222	0.837210	-1.332947	1.213162
<b>2022-01-09</b>	0.819371	-0.295471	-0.174542	-0.396840	-2.026224
<b>2022-01-10</b>	1.218235	-1.625886	-0.080381	1.723237	-0.365595

```
In [182... df.dtypes
```

```
Out[182... F    float64
A    float64
H    float64
A    float64
```

```
D      float64  
dtype: object
```

In [183...]

```
df1 = pd.DataFrame(  
    {  
        "A": 42.0,  
        "B": pd.date_range("20220101", periods=5),  
        "C": pd.Series(1, index=list(range(5)), dtype="float32"),  
        "D": np.array([2] * 5, dtype="int32"),  
        "E": pd.Categorical(["Male", "Female", "Male", "Female", "Male"]),  
        "F": "Human",  
        "G": np.array([1,2,3,5,6]),  
        "H": pd.Timestamp("20220131")  
    }  
)  
df1
```

Out[183...]

	A	B	C	D	E	F	G	H
0	42.0	2022-01-01	1.0	2	Male	Human	1	2022-01-31
1	42.0	2022-01-02	1.0	2	Female	Human	2	2022-01-31
2	42.0	2022-01-03	1.0	2	Male	Human	3	2022-01-31
3	42.0	2022-01-04	1.0	2	Female	Human	5	2022-01-31
4	42.0	2022-01-05	1.0	2	Male	Human	6	2022-01-31

In [184...]

```
df1.dtypes
```

Out[184...]

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

In [185...]

```
df1.head(2)
```

Out[185...]

	A	B	C	D	E	F	G	H
0	42.0	2022-01-01	1.0	2	Male	Human	1	2022-01-31
1	42.0	2022-01-02	1.0	2	Female	Human	2	2022-01-31

In [186...]

df1.tail(2)

Out[186...]

	A	B	C	D	E	F	G	H
3	42.0	2022-01-04	1.0	2	Female	Human	5	2022-01-31
4	42.0	2022-01-05	1.0	2	Male	Human	6	2022-01-31

In [187...]

df1.index

Out[187...]

Int64Index([0, 1, 2, 3, 4], dtype='int64')

In [188...]

df.index

Out[188...]

```
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 [189...]

df.to\_numpy

Out[189...]

```
<bound method DataFrame.to_numpy of
2022-01-01  0.834394  0.474209  1.677339 -1.494793 -1.157686
2022-01-02 -1.140119 -1.162909 -0.154563 -1.983856  0.497716
2022-01-03  2.223851  0.763401 -1.298154  1.100876  1.277980
2022-01-04 -1.848449 -0.287943 -0.333419  0.006742 -0.001046
2022-01-05 -0.194993 -0.323621 -0.400696  1.642857 -0.179146
2022-01-06 -1.176169 -0.527270 -0.095128  1.321913 -0.707781
2022-01-07  0.102521 -0.339865 -1.176499  1.476894  0.665398
2022-01-08 -0.546442  1.204222  0.837210 -1.332947  1.213162
2022-01-09  0.819371 -0.295471 -0.174542 -0.396840 -2.026224
2022-01-10  1.218235 -1.625886 -0.080381  1.723237 -0.365595>
```

In [190...]

```
df1.to_numpy
```

```
Out[190... <bound method DataFrame.to_numpy of
 0 42.0 2022-01-01 1.0 2 Male Human 1 2022-01-31
 1 42.0 2022-01-02 1.0 2 Female Human 2 2022-01-31
 2 42.0 2022-01-03 1.0 2 Male Human 3 2022-01-31
 3 42.0 2022-01-04 1.0 2 Female Human 5 2022-01-31
 4 42.0 2022-01-05 1.0 2 Male Human 6 2022-01-31>
```

```
In [191... df.describe()
```

```
Out[191...   F          A          H          A          D
count 10.000000 10.000000 10.000000 10.000000 10.000000
mean  0.029220 -0.212113 -0.119883  0.206408 -0.078322
std   1.257678  0.848018  0.868051  1.433234  1.045979
min   -1.848449 -1.625886 -1.298154 -1.983856 -2.026224
25%   -0.991699 -0.480418 -0.383876 -1.098920 -0.622235
50%   -0.046236 -0.309546 -0.164553  0.553809 -0.090096
75%   0.830638  0.283671 -0.084067  1.438149  0.623477
max   2.223851  1.204222  1.677339  1.723237  1.277980
```

```
In [192... df1.describe()
```

```
Out[192...   A          C          D          G
count  5.0        5.0        5.0      5.000000
mean  42.0       1.0        2.0      3.400000
std   0.0        0.0        0.0      2.073644
min   42.0       1.0        2.0      1.000000
25%   42.0       1.0        2.0      2.000000
50%   42.0       1.0        2.0      3.000000
```

	A	C	D	G
75%	42.0	1.0	2.0	5.000000
max	42.0	1.0	2.0	6.000000

In [193...]

```
# Transpose of Dataset !
df1.T
```

Out[193...]

	0	1	2	3	4
A	42.0	42.0	42.0	42.0	42.0
B	2022-01-01 00:00:00	2022-01-02 00:00:00	2022-01-03 00:00:00	2022-01-04 00:00:00	2022-01-05 00:00:00
C	1.0	1.0	1.0	1.0	1.0
D	2	2	2	2	2
E	Male	Female	Male	Female	Male
F	Human	Human	Human	Human	Human
G	1	2	3	5	6
H	2022-01-31 00:00:00	2022-01-31 00:00:00	2022-01-31 00:00:00	2022-01-31 00:00:00	2022-01-31 00:00:00

In [194...]

```
df1
```

Out[194...]

	A	B	C	D	E	F	G	H
0	42.0	2022-01-01	1.0	2	Male	Human	1	2022-01-31
1	42.0	2022-01-02	1.0	2	Female	Human	2	2022-01-31
2	42.0	2022-01-03	1.0	2	Male	Human	3	2022-01-31
3	42.0	2022-01-04	1.0	2	Female	Human	5	2022-01-31
4	42.0	2022-01-05	1.0	2	Male	Human	6	2022-01-31

## Sorting

In [195...]

```
# FALSE contains Descending output !
df1.sort_index(axis=0, ascending=False)
```

Out[195...]

	A	B	C	D	E	F	G	H
4	42.0	2022-01-05	1.0	2	Male	Human	6	2022-01-31
3	42.0	2022-01-04	1.0	2	Female	Human	5	2022-01-31
2	42.0	2022-01-03	1.0	2	Male	Human	3	2022-01-31
1	42.0	2022-01-02	1.0	2	Female	Human	2	2022-01-31
0	42.0	2022-01-01	1.0	2	Male	Human	1	2022-01-31

In [196...]

```
# TRUE contains Ascending output !
df1.sort_index(axis=0, ascending=True)
```

Out[196...]

	A	B	C	D	E	F	G	H
0	42.0	2022-01-01	1.0	2	Male	Human	1	2022-01-31
1	42.0	2022-01-02	1.0	2	Female	Human	2	2022-01-31
2	42.0	2022-01-03	1.0	2	Male	Human	3	2022-01-31
3	42.0	2022-01-04	1.0	2	Female	Human	5	2022-01-31
4	42.0	2022-01-05	1.0	2	Male	Human	6	2022-01-31

In [197...]

```
df1["E"] # Specific Columns
```

Out[197...]

```
0      Male
1    Female
2      Male
3    Female
4      Male
Name: E, dtype: category
Categories (2, object): ['Female', 'Male']
```

In [198...]

```
df
```

Out[198...]

	<b>F</b>	<b>A</b>	<b>H</b>	<b>A</b>	<b>D</b>
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716
<b>2022-01-03</b>	2.223851	0.763401	-1.298154	1.100876	1.277980
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	0.006742	-0.001046
<b>2022-01-05</b>	-0.194993	-0.323621	-0.400696	1.642857	-0.179146
<b>2022-01-06</b>	-1.176169	-0.527270	-0.095128	1.321913	-0.707781
<b>2022-01-07</b>	0.102521	-0.339865	-1.176499	1.476894	0.665398
<b>2022-01-08</b>	-0.546442	1.204222	0.837210	-1.332947	1.213162
<b>2022-01-09</b>	0.819371	-0.295471	-0.174542	-0.396840	-2.026224
<b>2022-01-10</b>	1.218235	-1.625886	-0.080381	1.723237	-0.365595

In [199...]

```
# sort by specific value !
df.sort_values(by="F")
```

Out[199...]

	<b>F</b>	<b>A</b>	<b>H</b>	<b>A</b>	<b>D</b>
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	0.006742	-0.001046
<b>2022-01-06</b>	-1.176169	-0.527270	-0.095128	1.321913	-0.707781
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716
<b>2022-01-08</b>	-0.546442	1.204222	0.837210	-1.332947	1.213162
<b>2022-01-05</b>	-0.194993	-0.323621	-0.400696	1.642857	-0.179146
<b>2022-01-07</b>	0.102521	-0.339865	-1.176499	1.476894	0.665398
<b>2022-01-09</b>	0.819371	-0.295471	-0.174542	-0.396840	-2.026224
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686
<b>2022-01-10</b>	1.218235	-1.625886	-0.080381	1.723237	-0.365595
<b>2022-01-03</b>	2.223851	0.763401	-1.298154	1.100876	1.277980

```
In [200... df["D"].head()
```

```
Out[200... 2022-01-01 -1.157686  
2022-01-02 0.497716  
2022-01-03 1.277980  
2022-01-04 -0.001046  
2022-01-05 -0.179146
```

Freq: D, Name: D, dtype: float64

```
In [201... df[0:]
```

	<b>F</b>	<b>A</b>	<b>H</b>	<b>A</b>	<b>D</b>
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716
<b>2022-01-03</b>	2.223851	0.763401	-1.298154	1.100876	1.277980
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	0.006742	-0.001046
<b>2022-01-05</b>	-0.194993	-0.323621	-0.400696	1.642857	-0.179146
<b>2022-01-06</b>	-1.176169	-0.527270	-0.095128	1.321913	-0.707781
<b>2022-01-07</b>	0.102521	-0.339865	-1.176499	1.476894	0.665398
<b>2022-01-08</b>	-0.546442	1.204222	0.837210	-1.332947	1.213162
<b>2022-01-09</b>	0.819371	-0.295471	-0.174542	-0.396840	-2.026224
<b>2022-01-10</b>	1.218235	-1.625886	-0.080381	1.723237	-0.365595

```
In [202... df[:3]
```

	<b>F</b>	<b>A</b>	<b>H</b>	<b>A</b>	<b>D</b>
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716
<b>2022-01-03</b>	2.223851	0.763401	-1.298154	1.100876	1.277980

```
In [203... df[0:5]
```

Out[203...]

	F	A	H	A	D
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716
<b>2022-01-03</b>	2.223851	0.763401	-1.298154	1.100876	1.277980
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	0.006742	-0.001046
<b>2022-01-05</b>	-0.194993	-0.323621	-0.400696	1.642857	-0.179146

In [204...]

df1[:3]

Out[204...]

	A	B	C	D	E	F	G	H
<b>0</b>	42.0	2022-01-01	1.0	2	Male	Human	1	2022-01-31
<b>1</b>	42.0	2022-01-02	1.0	2	Female	Human	2	2022-01-31
<b>2</b>	42.0	2022-01-03	1.0	2	Male	Human	3	2022-01-31

In [205...]

# row-wise selection !  
df.loc[dates[0]]

Out[205...]

F	0.834394
A	0.474209
H	1.677339
A	-1.494793
D	-1.157686
Name:	2022-01-01 00:00:00, dtype: float64

In [206...]

df.loc[dates[6]]

Out[206...]

F	0.102521
A	-0.339865
H	-1.176499
A	1.476894
D	0.665398
Name:	2022-01-07 00:00:00, dtype: float64

```
In [207... df.loc[dates[2:3]]
```

```
Out[207...      F      A      H      A      D  
2022-01-03  2.223851  0.763401 -1.298154  1.100876  1.27798
```

```
In [208... df.loc[[ "20220106", "20220108"], ["H", "D"]]
```

```
Out[208...      H      D  
2022-01-06 -0.095128 -0.707781  
2022-01-08  0.837210  1.213162
```

```
In [209... df.loc["20220106": "20220108", ["H", "D", "F"]]
```

```
Out[209...      H      D      F  
2022-01-06 -0.095128 -0.707781 -1.176169  
2022-01-07 -1.176499  0.665398  0.102521  
2022-01-08  0.837210  1.213162 -0.546442
```

```
In [210... df.loc["20220103", ["H", "D", "F"]]
```

```
Out[210... H    -1.298154  
D     1.277980  
F     2.223851  
Name: 2022-01-03 00:00:00, dtype: float64
```

```
In [211... df.loc["20220103"]
```

```
Out[211... F    2.223851  
A     0.763401  
H    -1.298154  
A     1.100876  
D     1.277980  
Name: 2022-01-03 00:00:00, dtype: float64
```

```
In [212... # at this specific date !
df.at[dates[5], "H"]
```

```
Out[212... -0.09512750274228017
```

```
In [213... df.iloc[0:2, [2,3]]
```

```
Out[213...      H      A
2022-01-01  1.677339 -1.494793
2022-01-02 -0.154563 -1.983856
```

```
In [214... df.iloc[2:6, 3:5]
```

```
Out[214...      A      D
2022-01-03  1.100876  1.277980
2022-01-04  0.006742 -0.001046
2022-01-05  1.642857 -0.179146
2022-01-06  1.321913 -0.707781
```

```
In [215... df.iloc[:2, 1:3]
```

```
Out[215...      A      H
2022-01-01  0.474209  1.677339
2022-01-02 -1.162909 -0.154563
```

```
In [216... df.iloc[2:4, :4]
```

```
Out[216...      F      A      H      A
2022-01-03  2.223851  0.763401 -1.298154  1.100876
```

	F	A	H	A
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	0.006742

In [217...]

```
df.head(2)
```

Out[217...]

	F	A	H	A	D
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716

In [218...]

```
df[df["H"] > 1]
```

Out[218...]

	F	A	H	A	D
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686

In [219...]

```
df[df < 0]
```

Out[219...]

	F	A	H	A	D
<b>2022-01-01</b>	NaN	NaN	NaN	-1.494793	-1.157686
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	NaN
<b>2022-01-03</b>	NaN	NaN	-1.298154	NaN	NaN
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	NaN	-0.001046
<b>2022-01-05</b>	-0.194993	-0.323621	-0.400696	NaN	-0.179146
<b>2022-01-06</b>	-1.176169	-0.527270	-0.095128	NaN	-0.707781
<b>2022-01-07</b>	NaN	-0.339865	-1.176499	NaN	NaN
<b>2022-01-08</b>	-0.546442	NaN	NaN	-1.332947	NaN
<b>2022-01-09</b>	NaN	-0.295471	-0.174542	-0.396840	-2.026224
<b>2022-01-10</b>	NaN	-1.625886	-0.080381	NaN	-0.365595

In [220...]

```
df1 = df.copy()
df1
```

Out[220...]

	<b>F</b>	<b>A</b>	<b>H</b>	<b>A</b>	<b>D</b>
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716
<b>2022-01-03</b>	2.223851	0.763401	-1.298154	1.100876	1.277980
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	0.006742	-0.001046
<b>2022-01-05</b>	-0.194993	-0.323621	-0.400696	1.642857	-0.179146
<b>2022-01-06</b>	-1.176169	-0.527270	-0.095128	1.321913	-0.707781
<b>2022-01-07</b>	0.102521	-0.339865	-1.176499	1.476894	0.665398
<b>2022-01-08</b>	-0.546442	1.204222	0.837210	-1.332947	1.213162
<b>2022-01-09</b>	0.819371	-0.295471	-0.174542	-0.396840	-2.026224
<b>2022-01-10</b>	1.218235	-1.625886	-0.080381	1.723237	-0.365595

In [221...]

```
df1["new"] = ["a", "b", "c", "d","a", "b", "c", "d","e","f"]
df1
```

Out[221...]

	<b>F</b>	<b>A</b>	<b>H</b>	<b>A</b>	<b>D</b>	<b>new</b>
<b>2022-01-01</b>	0.834394	0.474209	1.677339	-1.494793	-1.157686	a
<b>2022-01-02</b>	-1.140119	-1.162909	-0.154563	-1.983856	0.497716	b
<b>2022-01-03</b>	2.223851	0.763401	-1.298154	1.100876	1.277980	c
<b>2022-01-04</b>	-1.848449	-0.287943	-0.333419	0.006742	-0.001046	d
<b>2022-01-05</b>	-0.194993	-0.323621	-0.400696	1.642857	-0.179146	a
<b>2022-01-06</b>	-1.176169	-0.527270	-0.095128	1.321913	-0.707781	b
<b>2022-01-07</b>	0.102521	-0.339865	-1.176499	1.476894	0.665398	c
<b>2022-01-08</b>	-0.546442	1.204222	0.837210	-1.332947	1.213162	d

	F	A	H	A	D	new
2022-01-09	0.819371	-0.295471	-0.174542	-0.396840	-2.026224	e
2022-01-10	1.218235	-1.625886	-0.080381	1.723237	-0.365595	f

In [222...]

```
df1["mean"] = [1,2,3,4,5,2,3,4,5,1]
df1
```

Out[222...]

	F	A	H	A	D	new	mean
2022-01-01	0.834394	0.474209	1.677339	-1.494793	-1.157686	a	1
2022-01-02	-1.140119	-1.162909	-0.154563	-1.983856	0.497716	b	2
2022-01-03	2.223851	0.763401	-1.298154	1.100876	1.277980	c	3
2022-01-04	-1.848449	-0.287943	-0.333419	0.006742	-0.001046	d	4
2022-01-05	-0.194993	-0.323621	-0.400696	1.642857	-0.179146	a	5
2022-01-06	-1.176169	-0.527270	-0.095128	1.321913	-0.707781	b	2
2022-01-07	0.102521	-0.339865	-1.176499	1.476894	0.665398	c	3
2022-01-08	-0.546442	1.204222	0.837210	-1.332947	1.213162	d	4
2022-01-09	0.819371	-0.295471	-0.174542	-0.396840	-2.026224	e	5
2022-01-10	1.218235	-1.625886	-0.080381	1.723237	-0.365595	f	1

In [223...]

```
df1 = df1.iloc[2:5, 5:7] # Near Specific Locate !
```

In [224...]

```
df1
```

Out[224...]

	new	mean
2022-01-03	c	3
2022-01-04	d	4
2022-01-05	a	5

## Case Study with PANDAS

In [225...]

```
# import Libraires !

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

In [226...]

```
boat = sns.load_dataset("titanic")
boat.head(5)
```

Out[226...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [227...]

```
# Saving csv file !
boat.to_csv("boat.csv")
```

In [228...]

```
boat.describe()
```

Out[228...]

	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

	<b>survived</b>	<b>pclass</b>	<b>age</b>	<b>sibsp</b>	<b>parch</b>	<b>fare</b>
<b>50%</b>	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
<b>75%</b>	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
<b>max</b>	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [229...]

`boat.mean()`

```
C:\Users\fahad\AppData\Local\Temp\ipykernel_11712\416833424.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.
```

Out[229...]

```
boat.mean()
survived      0.383838
pclass        2.308642
age           29.699118
sibsp         0.523008
parch         0.381594
fare          32.204208
adult_male    0.602694
alone         0.602694
dtype: float64
```

In [230...]

`boat.head(2)`

Out[230...]

	<b>survived</b>	<b>pclass</b>	<b>sex</b>	<b>age</b>	<b>sibsp</b>	<b>parch</b>	<b>fare</b>	<b>embarked</b>	<b>class</b>	<b>who</b>	<b>adult_male</b>	<b>deck</b>	<b>embark_town</b>	<b>alive</b>	<b>alone</b>
<b>0</b>	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
<b>1</b>	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False

In [231...]

```
# Drop Few Columns from Dataset !
new_boat = boat.drop(["deck", "embark_town", "adult_male"], axis=1)
new_boat
```

Out[231...]

	<b>survived</b>	<b>pclass</b>	<b>sex</b>	<b>age</b>	<b>sibsp</b>	<b>parch</b>	<b>fare</b>	<b>embarked</b>	<b>class</b>	<b>who</b>	<b>alive</b>	<b>alone</b>
<b>0</b>	0	3	male	22.0	1	0	7.2500	S	Third	man	no	False
<b>1</b>	1	1	female	38.0	1	0	71.2833	C	First	woman	yes	False

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	alive	alone
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	no	True

891 rows × 12 columns

In [232...]

new\_boat.head(3)

Out[232...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	yes	True

In [233...]

boat.value\_counts(['survived'])

Out[233...]

survived	
0	549
1	342
	dtype: int64

In [234...]

boat.groupby(['sex']).mean()

Out[234...]

survived	pclass	age	sibsp	parch	fare	adult_male	alone
----------	--------	-----	-------	-------	------	------------	-------

sex	survived	pclass	age	sibsp	parch	fare	adult_male	alone
<b>sex</b>								
<b>female</b>	0.742038	2.159236	27.915709	0.694268	0.649682	44.479818	0.000000	0.401274
<b>male</b>	0.188908	2.389948	30.726645	0.429809	0.235702	25.523893	0.930676	0.712305

In [235...]

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

Out[235...]

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
sex	class							
<b>female</b>	<b>First</b>	0.968085	1.0	34.611765	0.553191	0.457447	106.125798	0.000000
	<b>Second</b>	0.921053	2.0	28.722973	0.486842	0.605263	21.970121	0.000000
	<b>Third</b>	0.500000	3.0	21.750000	0.895833	0.798611	16.118810	0.000000
<b>male</b>	<b>First</b>	0.368852	1.0	41.281386	0.311475	0.278689	67.226127	0.975410
	<b>Second</b>	0.157407	2.0	30.740707	0.342593	0.222222	19.741782	0.916667
	<b>Third</b>	0.135447	3.0	26.507589	0.498559	0.224784	12.661633	0.919308

In [236...]

```
boat[boat["age"] < 15].groupby(["sex"]).mean()
```

Out[236...]

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
sex								
<b>female</b>	0.615385	2.641026	6.410256	1.384615	1.282051	28.332159	0.0	0.076923
<b>male</b>	0.538462	2.615385	5.222308	2.256410	1.358974	35.076710	0.0	0.025641

In [237...]

```
boat[boat["age"] < 18].groupby(["sex", "class"]).mean()
```

Out[237...]

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
sex	class							
<b>female</b>	<b>First</b>	0.968085	1.0	34.611765	0.553191	0.457447	106.125798	0.000000
	<b>Second</b>	0.921053	2.0	28.722973	0.486842	0.605263	21.970121	0.000000

		survived	pclass	age	sibsp	parch	fare	adult_male	alone
sex	class								
<b>female</b>	<b>First</b>	0.875000	1.0	14.125000	0.500000	0.875000	104.083337	0.000000	0.125000
	<b>Second</b>	1.000000	2.0	8.333333	0.583333	1.083333	26.241667	0.000000	0.166667
	<b>Third</b>	0.542857	3.0	8.428571	1.571429	1.057143	18.727977	0.000000	0.228571
<b>male</b>	<b>First</b>	1.000000	1.0	8.230000	0.500000	2.000000	116.072900	0.250000	0.000000
	<b>Second</b>	0.818182	2.0	4.757273	0.727273	1.000000	25.659473	0.181818	0.181818
	<b>Third</b>	0.232558	3.0	9.963256	2.069767	1.000000	22.752523	0.348837	0.232558

## (Land) FAO Dataset

In [238...]

```
# import Libraries !
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [239...]

```
land = pd.read_csv("pakisatan_Land.csv")
land.head(3)
```

Out[239...]

	Domain Code	Domain	Area Code	Area	Element Code	Element	Item Code	Item	Year Code	Year	Unit	Value	Flag	Flag Description
<b>0</b>	RL	Land Use	165	Pakistan	5110	Area	6610	Agricultural land	1961	1961	1000 ha	35730.0	Fm	Manual Estimation
<b>1</b>	RL	Land Use	165	Pakistan	5110	Area	6610	Agricultural land	1962	1962	1000 ha	35840.0	Fm	Manual Estimation
<b>2</b>	RL	Land Use	165	Pakistan	5110	Area	6610	Agricultural land	1963	1963	1000 ha	35880.0	Fm	Manual Estimation

In [240...]

```
land.describe()
```

Out[240...]

	<b>Area Code</b>	<b>Element Code</b>	<b>Item Code</b>	<b>Year Code</b>	<b>Year</b>	<b>Value</b>
<b>count</b>	89.0	89.0	89.000000	89.000000	89.000000	89.000000
<b>mean</b>	165.0	5110.0	6630.224719	1994.887640	1994.887640	36375.441573
<b>std</b>	0.0	0.0	28.523417	16.355037	16.355037	673.361841
<b>min</b>	165.0	5110.0	6610.000000	1961.000000	1961.000000	35206.000000
<b>25%</b>	165.0	5110.0	6610.000000	1983.000000	1983.000000	35859.999000
<b>50%</b>	165.0	5110.0	6610.000000	1997.000000	1997.000000	36312.975000
<b>75%</b>	165.0	5110.0	6670.000000	2008.000000	2008.000000	36856.011000
<b>max</b>	165.0	5110.0	6670.000000	2019.000000	2019.000000	38509.000000

In [241...]

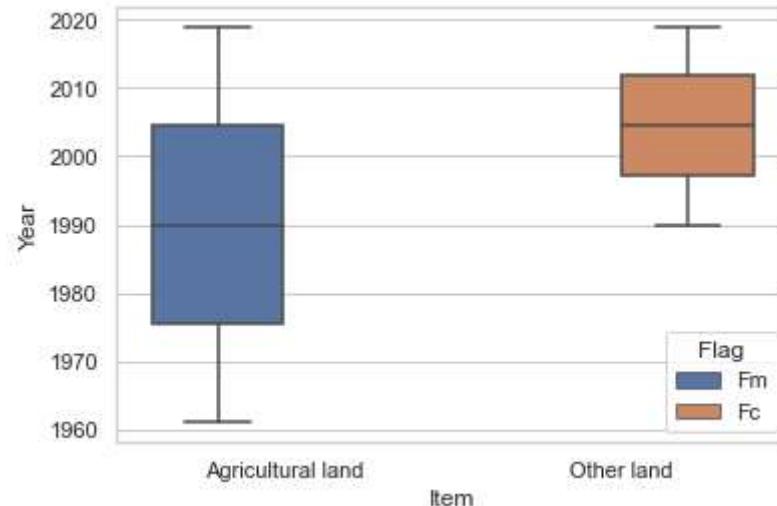
```
land.drop(["Area Code", "Element Code", "Element", "Domain", ], axis=1).head(3)
```

Out[241...]

	<b>Domain Code</b>	<b>Area</b>	<b>Item Code</b>	<b>Item</b>	<b>Year Code</b>	<b>Year</b>	<b>Unit</b>	<b>Value</b>	<b>Flag</b>	<b>Flag Description</b>
<b>0</b>	RL	Pakistan	6610	Agricultural land	1961	1961	1000 ha	35730.0	Fm	Manual Estimation
<b>1</b>	RL	Pakistan	6610	Agricultural land	1962	1962	1000 ha	35840.0	Fm	Manual Estimation
<b>2</b>	RL	Pakistan	6610	Agricultural land	1963	1963	1000 ha	35880.0	Fm	Manual Estimation

In [242...]

```
sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(x="Item",
             y="Year",
             hue="Flag",
             data=land)
plt.show()
```



In [243...]

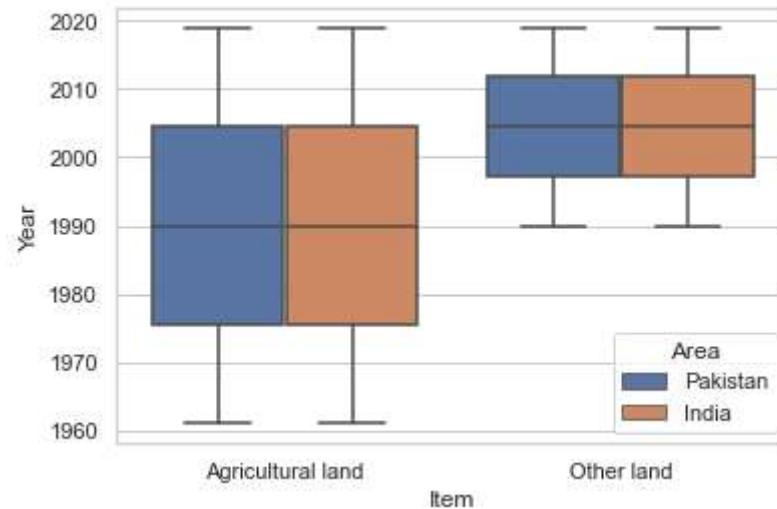
```
land = pd.read_csv("indo-Pak_Land.csv")
land.head(3)
```

Out[243...]

	Domain Code	Domain	Area Code	Area	Element Code	Element	Item Code	Item	Year Code	Year	Unit	Value	Flag	Flag Description
0	RL	Land Use	165	Pakistan	5110	Area	6610	Agricultural land	1961	1961	1000 ha	35730.0	Fm	Manual Estimation
1	RL	Land Use	165	Pakistan	5110	Area	6610	Agricultural land	1962	1962	1000 ha	35840.0	Fm	Manual Estimation
2	RL	Land Use	165	Pakistan	5110	Area	6610	Agricultural land	1963	1963	1000 ha	35880.0	Fm	Manual Estimation

In [244...]

```
sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(x="Item",
            y="Year",
            hue="Area",
            data=land)
plt.show()
```



## (6). Statistics

### EDA. (Exploratory Data Analysis)

In [245...]

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

In [246...]

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

In [247...]

```
boat.to_csv("titanic.csv")
```

## | 01\_Undestand the Data !

In [248...]

```
boat.head()
```

Out[248...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [249...]

boat.tail()

Out[249...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
886	0	2	male	27.0	0	0	13.00	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.00	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.45	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.00	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.75	Q	Third	man	True	NaN	Queenstown	no	True

In [250...]

boat.shape

Out[250...]

(891, 15)

In [251...]

boat.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   cabin       891 non-null    object 
 8   embarked    891 non-null    object 
 9   class        891 non-null    object 
 10  who         891 non-null    object 
 11  adult_male  891 non-null    bool    
 12  deck        891 non-null    object 
 13  embark_town 891 non-null    object 
 14  alive       891 non-null    bool    
 15  alone      891 non-null    bool    

```

```

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: 63.0+ KB

```

In [252...]

`boat.describe()`

Out[252...]

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

In [253...]

`boat.columns`

Out[253...]

```

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

```

In [254...]

`boat.nunique()`

Out[254...]

	<b>survived</b>	<b>2</b>
<b>pclass</b>	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 [255... boat['who'].unique()
```

```
Out[255... array(['man', 'woman', 'child'], dtype=object)
```

```
In [256... boat['class'].unique()
```

```
Out[256... ['Third', 'First', 'Second']
Categories (3, object): ['First', 'Second', 'Third']
```

```
In [257... boat['alone'].append(boat['alive']).unique()
```

```
Out[257... array([False, True, 'no', 'yes'], dtype=object)
```

## 02\_Cleaning & Filtering the Data !

### - Find missing values

```
In [258... boat.isnull().sum()
```

```
Out[258... 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
```

## -- Remove missing values

In [259...]

```
# Remove duck columns !
boat_new = boat.drop(['deck'], axis=1)
boat_new.head(2)
```

Out[259...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone
<b>0</b>	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Southampton	no	False
<b>1</b>	1	1	female	38.0	1	0	71.2833	C	First	woman	False	Cherbourg	yes	False

In [260...]

```
boat_new.columns
```

Out[260...]

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

In [261...]

```
boat_new.isnull().sum()
```

Out[261...]

```
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
```

## -- Remove missing values

In [262...]: boat\_new.shape

Out[262...]: (891, 14)

In [263...]: 891-177-2-2

Out[263...]: 710

In [264...]: boat\_new1 = boat\_new.dropna()  
boat\_new1.head()

Out[264...]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone
<b>0</b>	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Southampton	no	False
<b>1</b>	1	1	female	38.0	1	0	71.2833	C	First	woman	False	Cherbourg	yes	False
<b>2</b>	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Southampton	yes	True
<b>3</b>	1	1	female	35.0	1	0	53.1000	S	First	woman	False	Southampton	yes	False
<b>4</b>	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Southampton	no	True

In [265...]: boat\_new1.isnull().sum()

Out[265...]:

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 [266...]: boat\_new1.shape

Out[266...]: (712, 14)

In [267...]: boat\_new1['who'].value\_counts()

Out[267...]:

man	413
woman	216
child	83

Name: who, dtype: int64

In [268...]: boat\_new1['alive'].value\_counts()

Out[268...]:

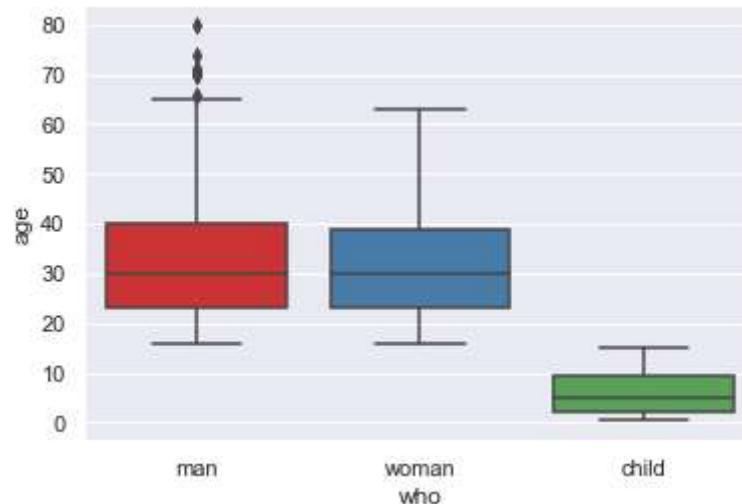
no	424
yes	288

Name: alive, dtype: int64

## -- Remove Outliers

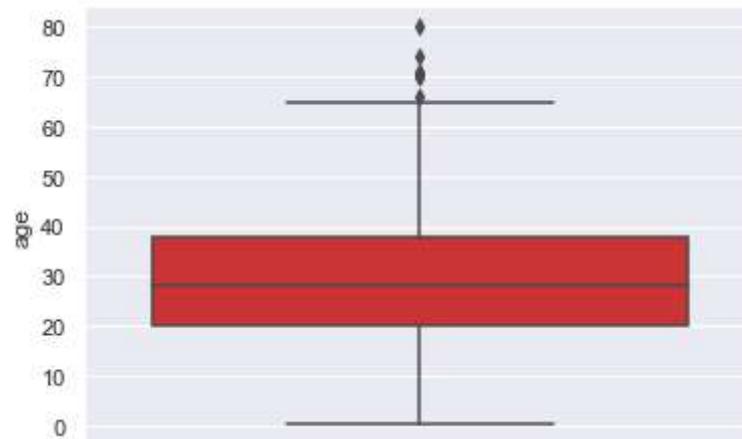
```
# box-plot !

sns.set_theme(style="darkgrid", color_codes=True)
sns.boxplot(data=boat_new1,
            x="who",
            y="age",
            palette="Set1")
plt.show()
```



In [270...]

```
# box-plot !  
  
sns.set_theme(style="darkgrid", color_codes=True)  
sns.boxplot(data=boat_new1,  
            y="age",  
            palette="Set1")  
plt.show()
```



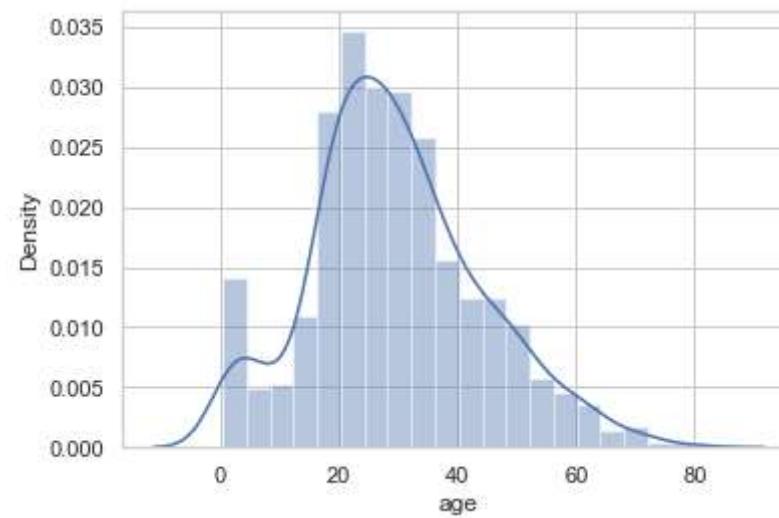
In [271...]

```
# dist-plot !  
  
sns.set_theme(style="whitegrid", color_codes=True)
```

```
sns.distplot(boat_new1['age'])  
plt.show()
```

c:\python38\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)
```



In [272...]

```
boat_new1.age
```

Out[272...]  
0 22.0  
1 38.0  
2 26.0  
3 35.0  
4 35.0  
...  
885 39.0  
886 27.0  
887 19.0  
889 26.0  
890 32.0

Name: age, Length: 712, dtype: float64

In [273...]

```
boat_new1['age'].mean()
```

Out[273...]  
29.64209269662921

In [274...]

boat\_new1.columns

Out[274...]

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

In [275...]

```
boat_new1 = boat_new1[boat_new1['age'] <= 60]
boat_new1.head(3)
```

Out[275...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	Southampton	yes	True

In [276...]

boat\_new1.age

Out[276...]

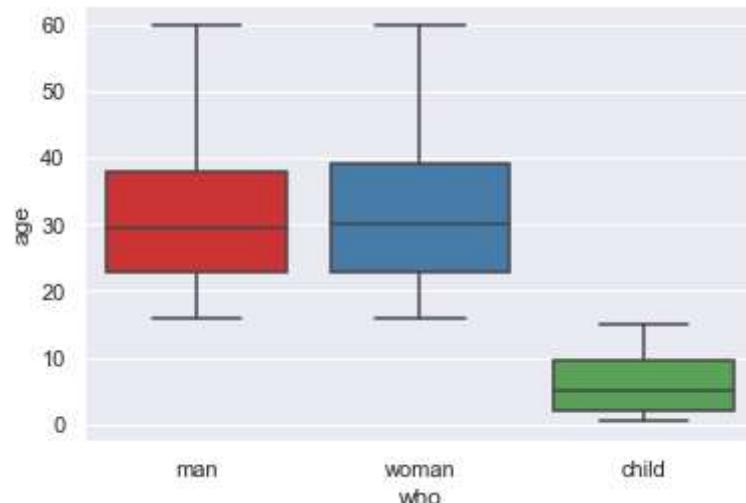
```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
...
885   39.0
886   27.0
887   19.0
889   26.0
890   32.0
Name: age, Length: 691, dtype: float64
```

## Plots

In [277...]

```
# box-plot !
sns.set_theme(style="darkgrid", color_codes=True)
sns.boxplot(data=boat_new1,
```

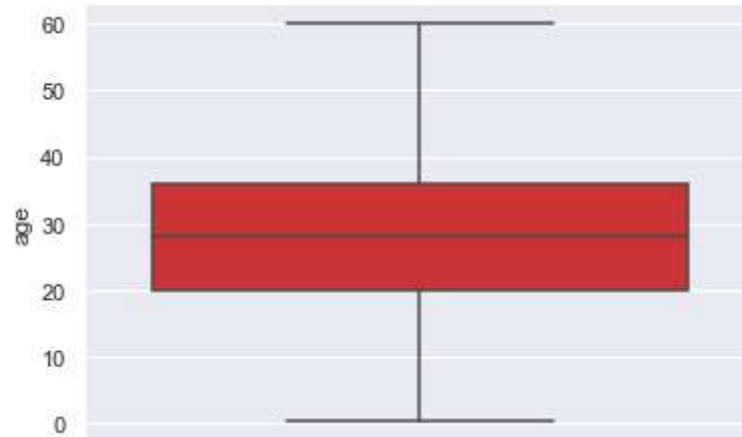
```
x="who",
y="age",
palette="Set1")
plt.show()
```



In [278...]

```
# box-plot ! of men

sns.set_theme(style="darkgrid", color_codes=True)
sns.boxplot(data=boat_new1,
            y="age",
            palette="Set1")
plt.show()
```

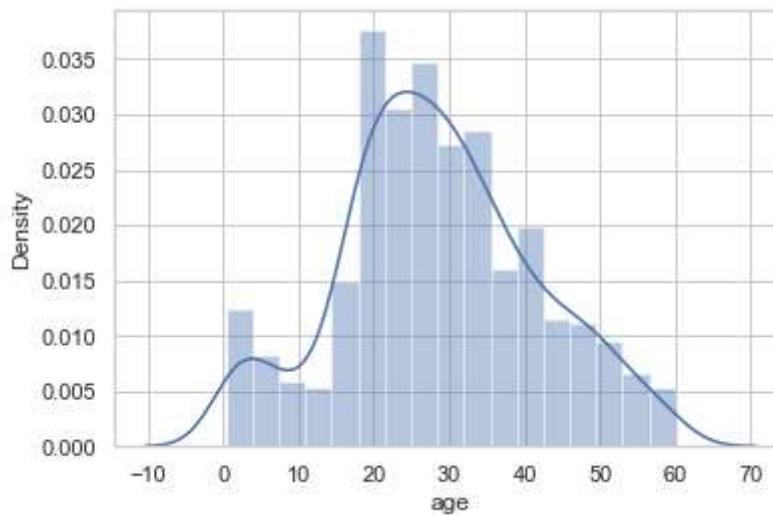


In [279...]

```
# dist-plot !  
  
sns.set_theme(style="whitegrid", color_codes=True)  
sns.distplot(boat_new1['age'])  
plt.show()
```

c:\python38\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).

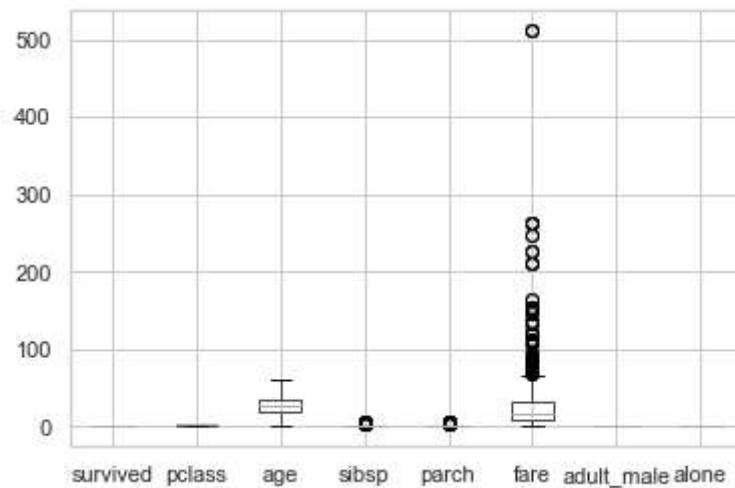
```
warnings.warn(msg, FutureWarning)
```



```
In [280... # box-plot !
```

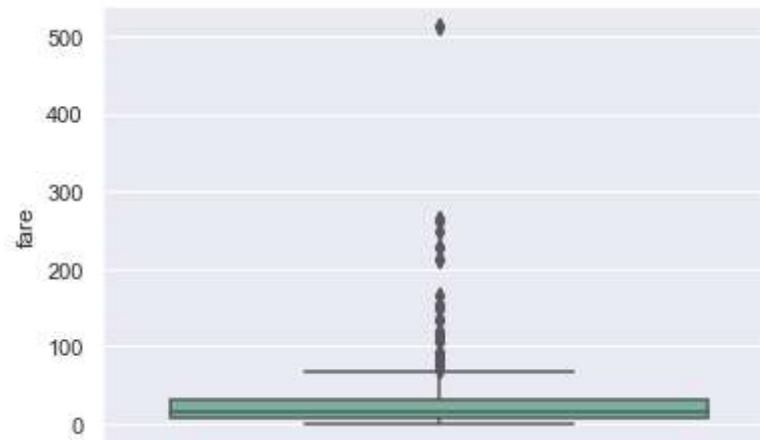
```
boat_new1.boxplot()
```

```
Out[280... <AxesSubplot:>
```



```
In [281... # box-plot ! of men
```

```
sns.set_theme(style="darkgrid", color_codes=True)
sns.boxplot(data=boat_new1,
            y="fare",
            palette="Set2")
plt.show()
```

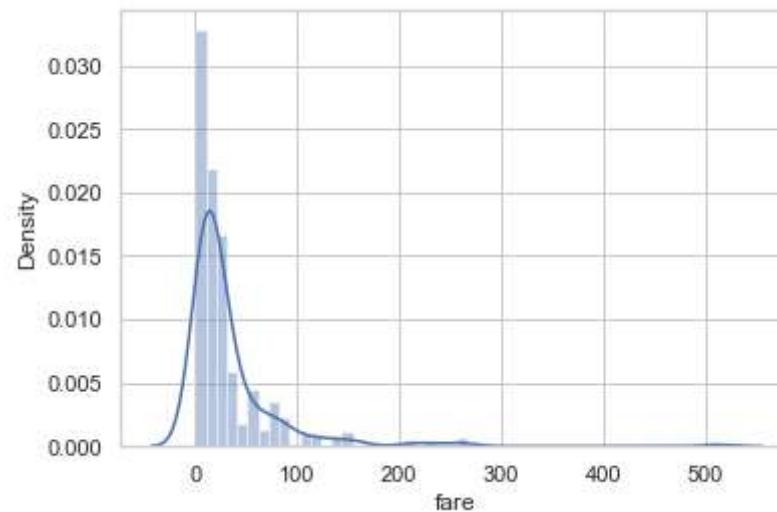


In [282...]

```
# dist-plot !  
  
sns.set_theme(style="whitegrid", color_codes=True)  
sns.distplot(boat_new1['fare'])  
plt.show()
```

c:\python38\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)
```



In [283...]

```
boat_new1['fare'].max()
```

```
Out[283... 512.3292
```

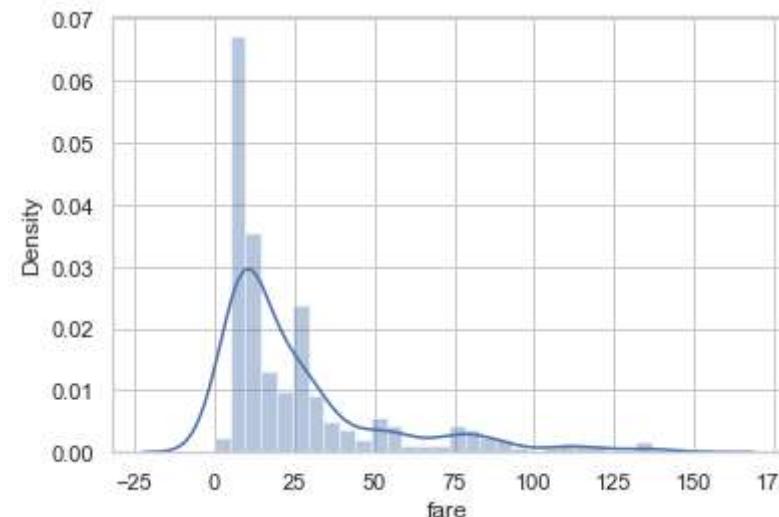
```
In [284... boat_new1 = boat_new1[boat_new1['fare']<=150]
boat_new1.fare.max()
```

```
Out[284... 146.5208
```

```
In [285... # dist-plot !
sns.set_theme(style="whitegrid", color_codes=True)
sns.distplot(boat_new1['fare'])
plt.show()
```

c:\python38\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)
```



```
In [286... # Log transformation !
```

```
boat_new1['log_fare'] = np.log(boat_new1[['fare']])
boat_new1.head(2)
```

Out[286...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone	log_fare
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Southampton	no	False	1.981001
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	Cherbourg	yes	False	4.266662

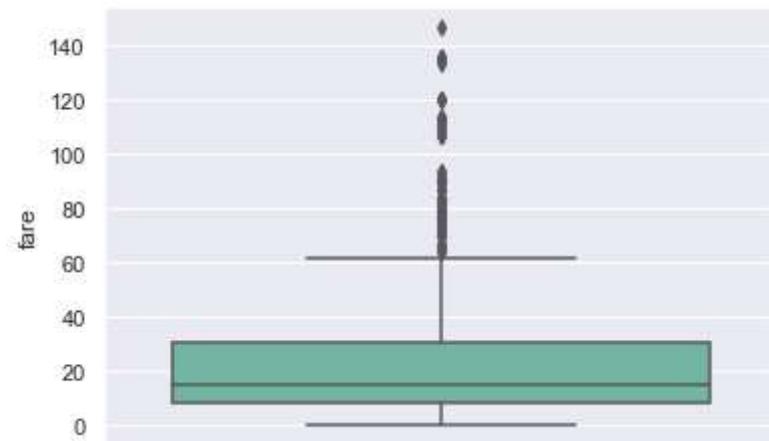
In [287...]

# box-plot ! of men

```

sns.set_theme(style="darkgrid", color_codes=True)
sns.boxplot(data=boat_new1,
            y="fare",
            palette="Set2")
plt.show()

```



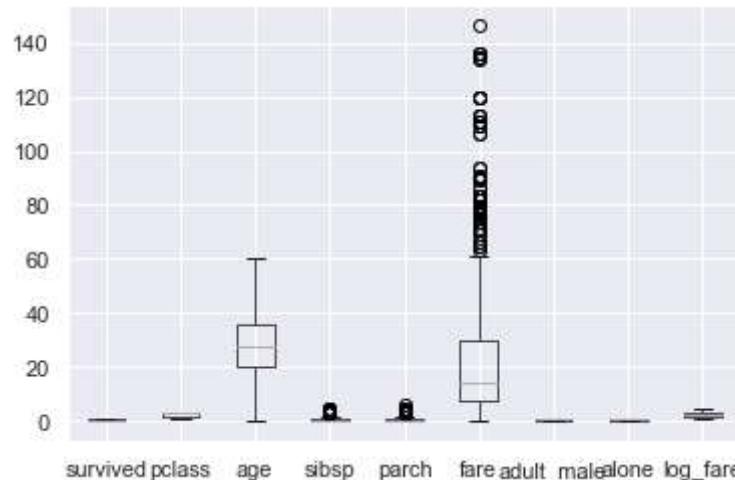
In [288...]

# box-plot !

boat\_new1.boxplot()

Out[288...]

&lt;AxesSubplot:&gt;



In [289...]

boat.shape

Out[289...]

(891, 15)

In [290...]

boat\_new1.shape

Out[290...]

(665, 15)

In [291...]

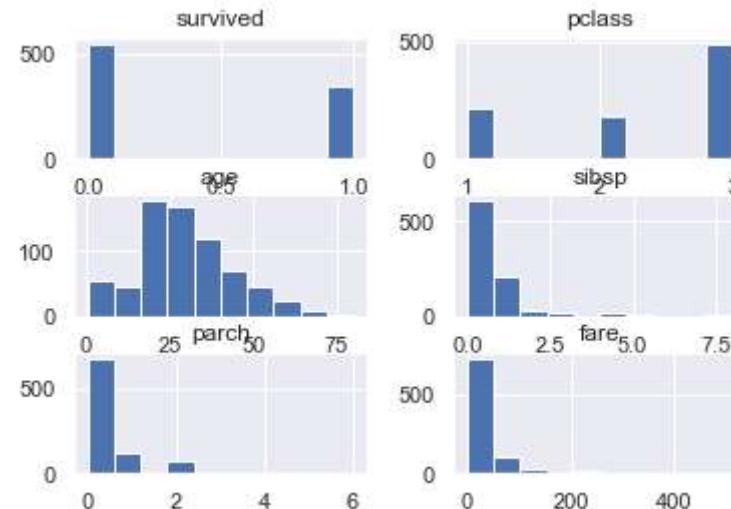
# (Log transformation) for Normalize our data !

In [292...]

boat.hist()

Out[292...]

array([[&lt;AxesSubplot:title={'center':'survived'}&gt;,&lt;AxesSubplot:title={'center':'pclass'}&gt;], [&lt;AxesSubplot:title={'center':'age'}&gt;,&lt;AxesSubplot:title={'center':'sibsp'}&gt;], [&lt;AxesSubplot:title={'center':'parch'}&gt;,&lt;AxesSubplot:title={'center':'fare'}&gt;]], dtype=object)

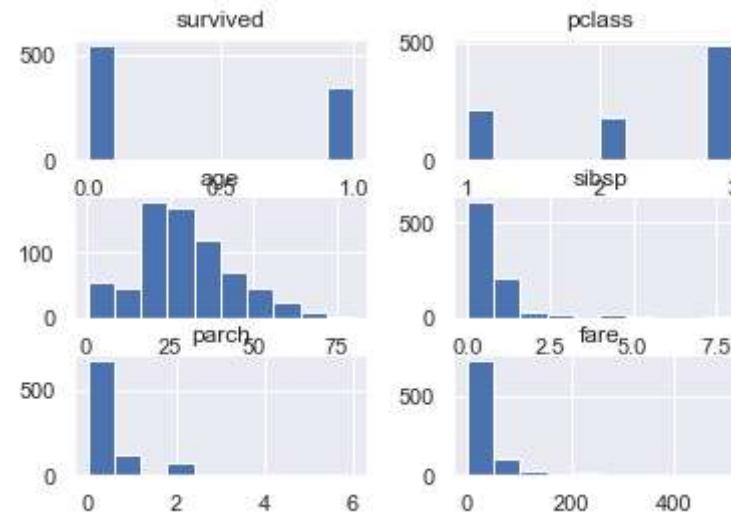


In [293...]

`boat_new.hist()`

Out[293...]

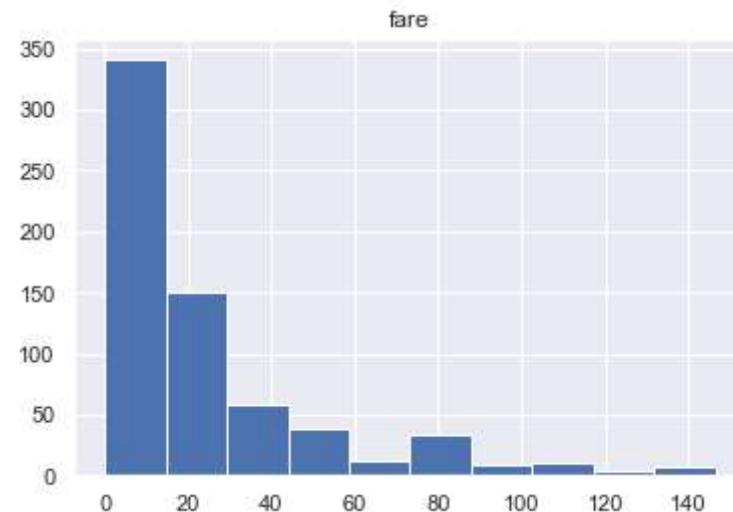
```
array([[<AxesSubplot:title={'center':'survived'}>,
       <AxesSubplot:title={'center':'pclass'}>],
      [<AxesSubplot:title={'center':'age'}>,
       <AxesSubplot:title={'center':'sibsp'}>],
      [<AxesSubplot:title={'center':'parch'}>,
       <AxesSubplot:title={'center':'fare'}>]], dtype=object)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



In [294...]

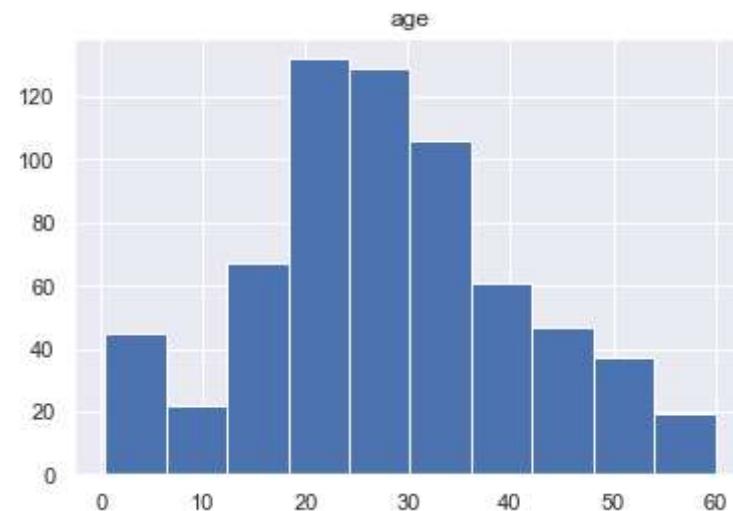
`boat_new1.hist(['fare'])`

```
Out[294... array([[<AxesSubplot:title={'center':'fare'}>]], dtype=object)
```



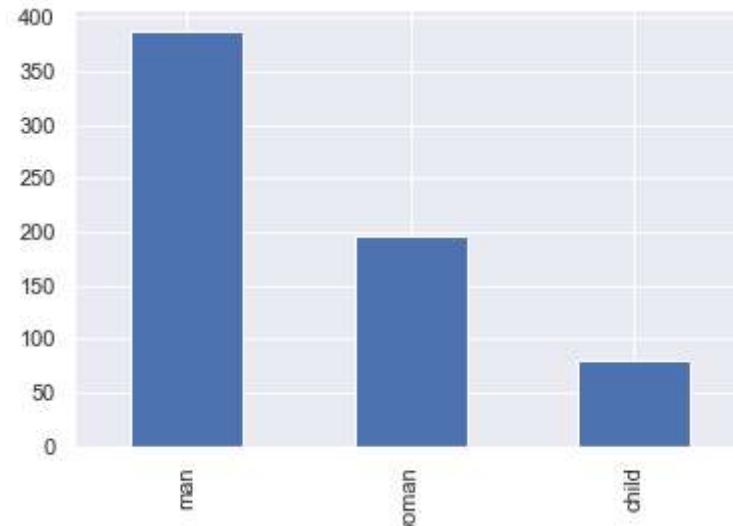
```
In [295... boat_new1.hist(['age'])
```

```
Out[295... array([[<AxesSubplot:title={'center':'age'}>]], dtype=object)
```



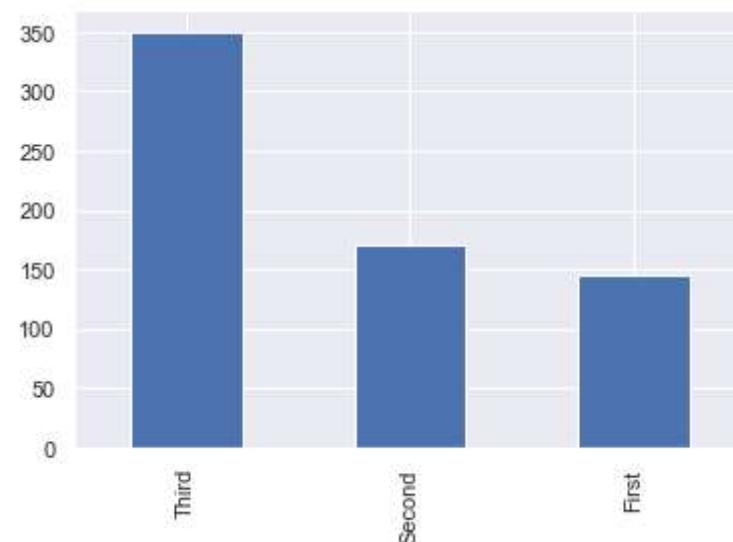
```
In [296... pd.value_counts(boat_new1['who']).plot.bar()
```

Out[296... &lt;AxesSubplot:&gt;



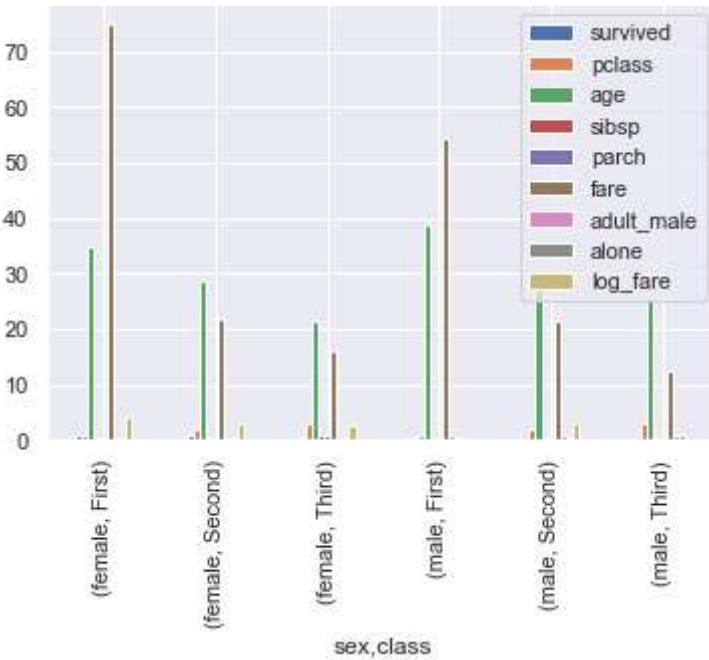
In [297... pd.value\_counts(boat\_new1['class']).plot.bar()

Out[297... &lt;AxesSubplot:&gt;



In [298... boat\_new1.groupby(['sex', 'class']).mean().plot.bar()

Out[298... &lt;AxesSubplot:xlabel='sex,class'&gt;



In [299... boat.groupby(['sex', 'class']).mean()

		survived	pclass	age	sibsp	parch	fare	adult_male	alone
sex	class								
female	First	0.968085	1.0	34.611765	0.553191	0.457447	106.125798	0.000000	0.361702
	Second	0.921053	2.0	28.722973	0.486842	0.605263	21.970121	0.000000	0.421053
	Third	0.500000	3.0	21.750000	0.895833	0.798611	16.118810	0.000000	0.416667
male	First	0.368852	1.0	41.281386	0.311475	0.278689	67.226127	0.975410	0.614754
	Second	0.157407	2.0	30.740707	0.342593	0.222222	19.741782	0.916667	0.666667
	Third	0.135447	3.0	26.507589	0.498559	0.224784	12.661633	0.919308	0.760807

In [300... boat\_new1.groupby(['sex', 'class']).mean()

Out[300...]

		survived	pclass	age	sibsp	parch	fare	adult_male	alone	log_fare
sex	class									
female	First	0.984127	1.0	34.920635	0.492063	0.365079	74.904432	0.00000	0.380952	4.221755
	Second	0.918919	2.0	28.722973	0.500000	0.621622	21.951070	0.00000	0.405405	2.985791
	Third	0.455446	3.0	21.341584	0.831683	0.960396	15.937625	0.00000	0.366337	2.621204
male	First	0.439024	1.0	38.945122	0.390244	0.231707	54.294156	0.97561	0.536585	NaN
	Second	0.145833	2.0	29.638854	0.385417	0.250000	21.444792	0.90625	0.625000	2.906213
	Third	0.152610	3.0	25.847068	0.497992	0.261044	12.239556	0.88755	0.734940	NaN

In [301...]

boat\_new1.isnull().sum()

Out[301...]

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
log_fare	0
dtype: int64	

## 03\_Relationship between Data !

In [302...]

```
boat_corr = boat.corr()
boat_corr
```

Out[302...]

	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307	-0.557080	-0.203367

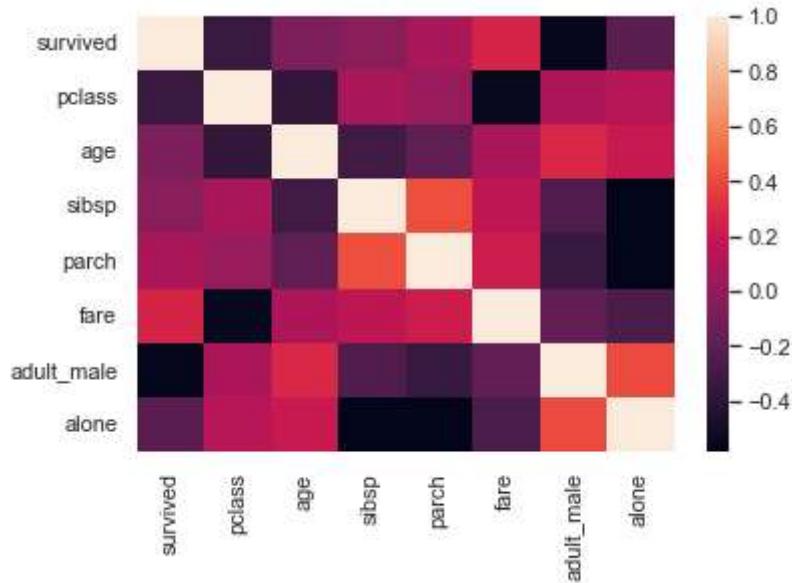
	<b>survived</b>	<b>pclass</b>	<b>age</b>	<b>sibsp</b>	<b>parch</b>	<b>fare</b>	<b>adult_male</b>	<b>alone</b>
<b>pclass</b>	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500	0.094035	0.135207
<b>age</b>	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067	0.280328	0.198270
<b>sibsp</b>	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651	-0.253586	-0.584471
<b>parch</b>	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225	-0.349943	-0.583398
<b>fare</b>	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000	-0.182024	-0.271832
<b>adult_male</b>	-0.557080	0.094035	0.280328	-0.253586	-0.349943	-0.182024	1.000000	0.404744
<b>alone</b>	-0.203367	0.135207	0.198270	-0.584471	-0.583398	-0.271832	0.404744	1.000000

In [303...]

`sns.heatmap(boat_corr)`

Out[303...]

&lt;AxesSubplot:&gt;

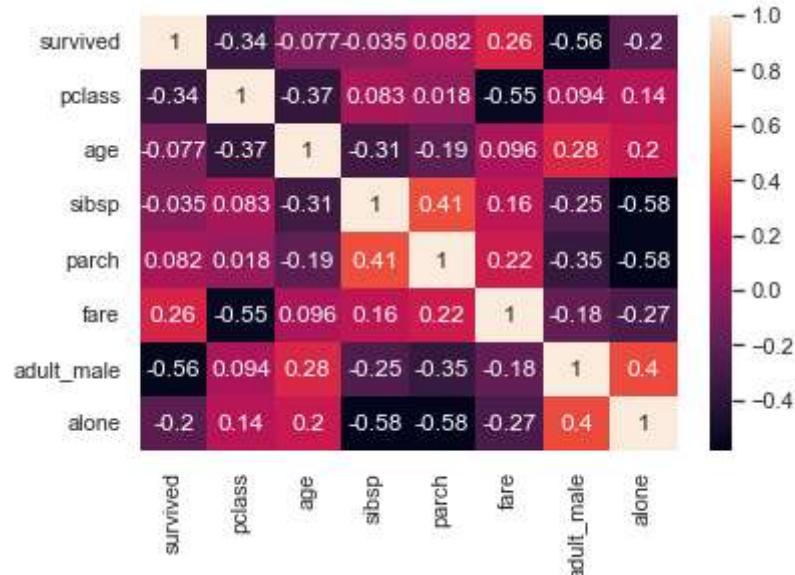


In [304...]

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

Out[304...]

&lt;AxesSubplot:&gt;

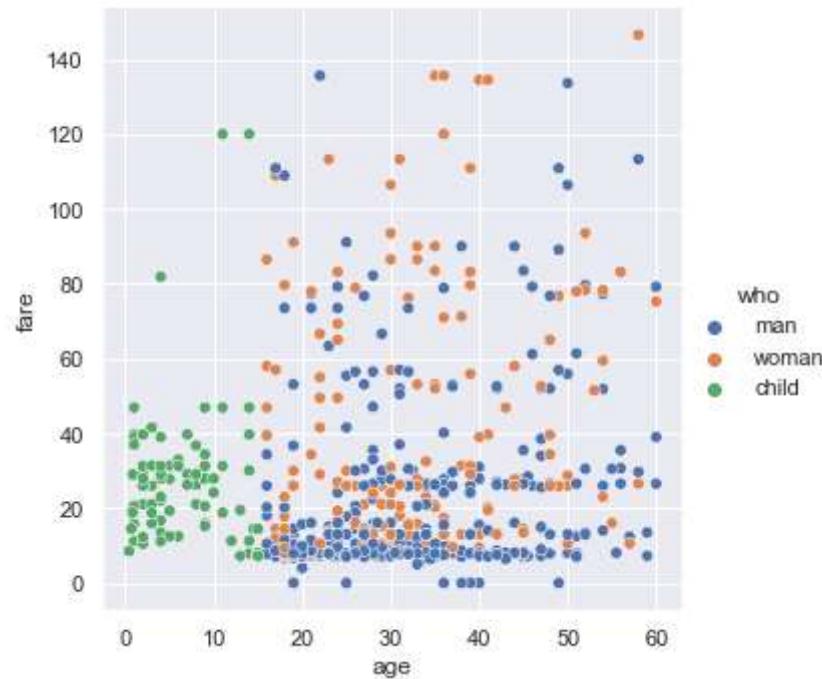


In [305...]

```
sns.relplot(x='age',
            y='fare',
            hue='who',
            data=boat_new1)
```

Out[305...]

&lt;seaborn.axisgrid.FacetGrid at 0x17483ac0&gt;

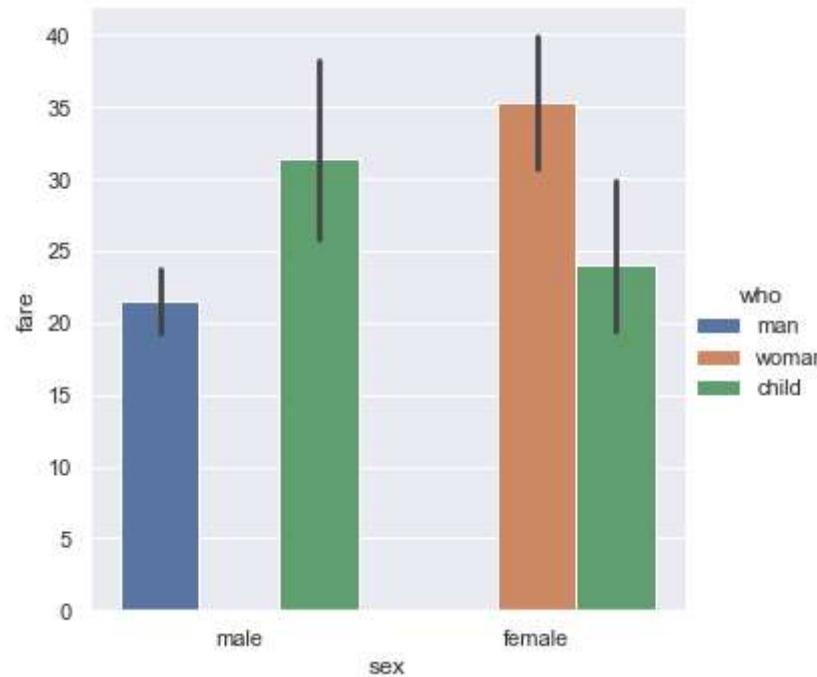


In [306...]

```
sns.catplot(x='sex',
             y='fare',
             hue='who',
             data=boat_new1,
             kind='bar')
```

Out[306...]

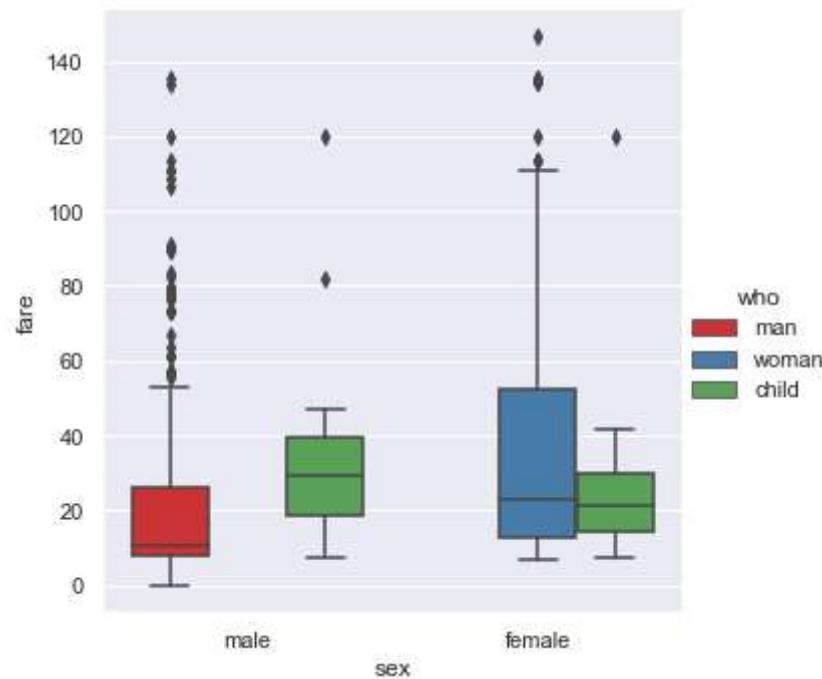
```
<seaborn.axisgrid.FacetGrid at 0x173d3fb8>
```



In [307...]

```
sns.catplot(x='sex',
             y='fare',
             hue='who',
             data=boat_new1,
             kind='box',
             palette="Set1")
```

Out[307...]



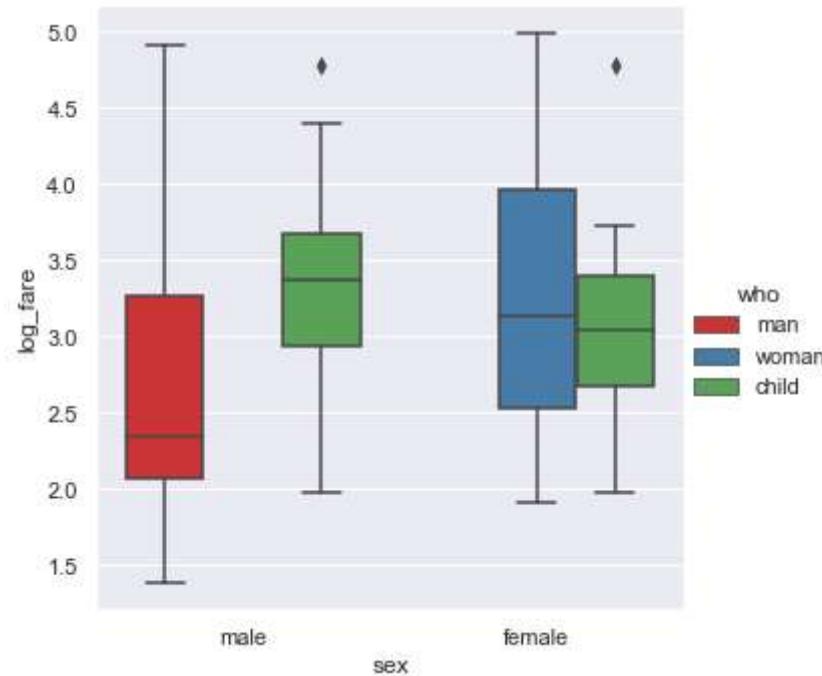
After taking log(fare)

In [308...]

```
sns.catplot(x='sex',
             y='log_fare',
             hue='who',
             data=boat_new1,
             kind='box',
             palette="Set1")
```

Out[308...]

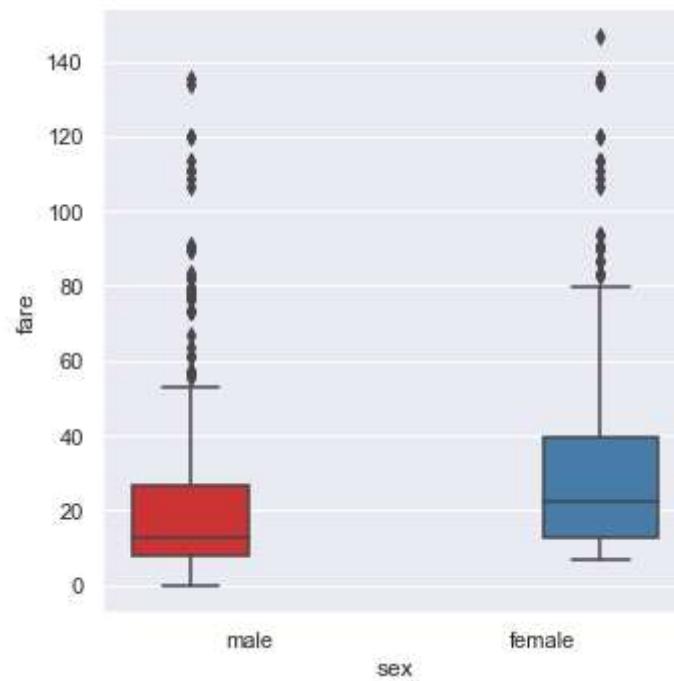
```
<seaborn.axisgrid.FacetGrid at 0x173ba610>
```



In [309...]

```
sns.catplot(x='sex',
             y='fare',
             hue='sex',
             data=boat_new1,
             kind='box',
             palette="Set1")
```

Out[309...]



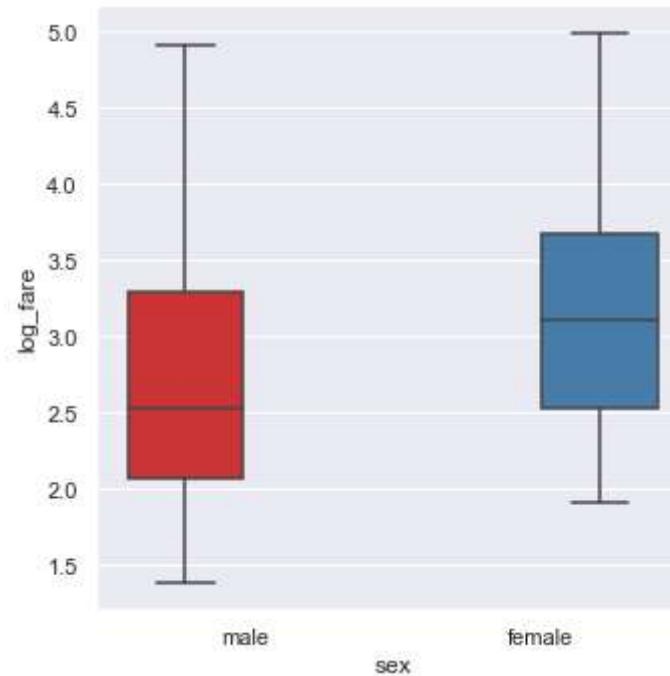
After taking log(fare)

In [310...]

```
sns.catplot(x='sex',
             y='log_fare',
             hue='sex',
             data=boat_new1,
             kind='box',
             palette="Set1")
```

Out[310...]

```
<seaborn.axisgrid.FacetGrid at 0x17d1f2b0>
```



### \*\*\*Data Wrangling\*\*\* ! (Data Pre-processing)

In [311...]

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

In [312...]

```
boat = sns.load_dataset('titanic')
boat1 = sns.load_dataset('titanic')
boat2 = sns.load_dataset('titanic')
```

In [313...]

```
boat.head()
```

Out[313...]

survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
----------	--------	-----	-----	-------	-------	------	----------	-------	-----	------------	------	-------------	-------	-------

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

### Simple Operation. (Math operator)

In [314...]

```
(boat['age']+1).head(3)
```

Out[314...]

```
0    23.0
1    39.0
2    27.0
Name: age, dtype: float64
```

### Find missing values

In [315...]

```
boat.isnull().sum()
```

Out[315...]

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
```

---

### ***Find shape of Dataset***

```
In [316... boat.shape
```

```
Out[316... (891, 15)
```

---

```
In [317... boat.deck.isnull().sum()
```

```
Out[317... 688
```

---

### ***Remove the missing values of specific column.(deck) in x-axis***

```
In [318... boat.deck.isnull().sum()
```

```
Out[318... 688
```

---

```
In [319... boat.dropna(subset=['deck'], axis=0, inplace=True)
```

```
In [320... boat.deck.isnull().sum()
```

```
Out[320... 0
```

---

```
In [321... boat.shape
```

```
Out[321... (203, 15)
```

---

```
In [322... boat.isnull().sum()
```

```
Out[322... 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
```

### ***Remove the missing values from whole Dataset***

```
In [323... boat = boat.dropna()  
boat.isnull().sum()
```

```
Out[323... 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
```

```
In [324... boat.shape
```

Out[324... (182, 15)

In [325... boat1.head(2)

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False

In [326... boat1.shape

Out[326... (891, 15)

In [327... boat1.isnull().sum()

```

Out[327... 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

```

### Find mean of age

In [328... age\_mean = boat1.age.mean()  
age\_mean

Out[328... 29.69911764705882

### Replace the missing values with the average(mean) of that column.(age)

In [329... boat1.age.isnull().sum()

Out[329... 177

In [330... boat1['age'] = boat1['age'].replace(np.nan, age\_mean)

In [331... boat1.age.isnull().sum()

Out[331... 0

### Remove the missing values of specific column.(deck) in x-axis

In [332... boat1.deck.isnull().sum()

Out[332... 688

In [333... boat1.dropna(subset=['deck'], axis=0, inplace=True)

In [334... boat1.deck.isnull().sum()

Out[334... 0

In [335... boat1.shape

Out[335... (203, 15)

## Remove the missing values from whole Dataset

In [336...]

```
boat1 = boat1.dropna()
boat1.isnull().sum()
```

Out[336...]

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	

In [337...]

```
boat1.shape
```

Out[337...]

(201, 15)	

## Data Cleaning !

### Data Formatting/ Duplicate Removal/ Data Standardization

In [338...]

```
boat2.head(2)
```

Out[338...]

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False

In [339...]

### boat2.dtypes

```
Out[339... boat2.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

### Convert data-types of indexes

```
In [340... boat2['survived'].dtypes
```

```
Out[340... dtype('int64')
```

```
In [341... boat2['survived'] = boat2['survived'].astype('float32')
```

```
In [342... boat2['survived'].dtypes
```

```
Out[342... dtype('float32')
```

### Convert age in days

```
In [343... boat2.age.head()
```

```
Out[343... 0    22.0
```

```
1    38.0
2    26.0
3    35.0
4    35.0
Name: age, dtype: float64
```

In [344...]: boat2['age'] = boat2['age']\*365

In [345...]: boat2.age.head()

```
0    8030.0
1   13870.0
2   9490.0
3  12775.0
4  12775.0
Name: age, dtype: float64
```

### Rename column (age to age\_days)

In [346...]: boat2.head(1)

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0.0	3	male	8030.0	1	0	7.25	S	Third	man	True	NaN	Southampton	no	False

In [347...]: boat2['age'] = boat2.rename(columns={'age': 'age\_days'}, inplace=True)

In [348...]: boat2.head(1)

	survived	pclass	sex	age_days	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone	age
0	0.0	3	male	8030.0	1	0	7.25	S	Third	man	True	NaN	Southampton	no	False	None

## Data Normalization / Human-Readable size /Machine-Readable size ! (Bell curve)

## Parametric Analysis tests

## Non-Parametric Analysis tests

### **Method of Normalize Data.** (Scaling Method)

- Simple Feature
- Min-Max Method
- Z-score. (Standard Score)
- log transformation

### Formula's

- *Simple Feature.* { `set['value'] = set['value'] / set['value'].max()` }
- *Min-Max Method.* { `set['value'] = (set['value']-set['value'].min()) / (set['value'].max()-set['value'].min())` }
- *Z-score. (Standard Score).* { `set['value'] = (set['value']-set['value'].mean()) / set['value'].std()` }
- *log transformation.* { `set['value'] = np.log(set['value'])` }

### Simple Feature Scaling

- Range.(0 to 1)

In [349...]

```
m1 = boat2[['age_days', 'fare']]
m1.head()
```

Out[349...]

	age_days	fare
0	8030.0	7.2500
1	13870.0	71.2833
2	9490.0	7.9250
3	12775.0	53.1000
4	12775.0	8.0500

In [350...]

```
m1['age_days'] = m1['age_days']/m1['age_days'].max()  
m1['fare'] = m1['fare']/m1['fare'].max()  
  
m1.head()
```

C:\Users\fahad\AppData\Local\Temp\ipykernel\_11712/1515070788.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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
m1['age_days'] = m1['age_days']/m1['age_days'].max()  
C:\Users\fahad\AppData\Local\Temp\ipykernel_11712/1515070788.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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
m1['fare'] = m1['fare']/m1['fare'].max()
```

Out[350...]

	age_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

In [351...]

```
m1.min()
```

Out[351...]

```
age_days    0.00525  
fare        0.00000  
dtype: float64
```

In [352...]

```
m1.max()
```

Out[352...]

```
age_days    1.0  
fare        1.0  
dtype: float64
```

## Min-Max Scaling Method

- Range.(0 to 1)

In [353...]

```
m2 = boat2[['age_days', 'fare']]
m2.head()
```

Out[353...]

	age_days	fare
0	8030.0	7.2500
1	13870.0	71.2833
2	9490.0	7.9250
3	12775.0	53.1000
4	12775.0	8.0500

In [354...]

```
m2['age_days'] = (m2['age_days']-m2['age_days'].min()) / (m2['age_days'].max()-m2['age_days'].min())
m2['fare'] = (m2['fare']-m2['fare'].min()) / (m2['fare'].max()-m2['fare'].min())
m2.head()
```

C:\Users\fahad\AppData\Local\Temp\ipykernel\_11712\2700239246.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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
m2['age_days'] = (m2['age_days']-m2['age_days'].min()) / (m2['age_days'].max()-m2['age_days'].min())
C:\Users\fahad\AppData\Local\Temp\ipykernel_11712\2700239246.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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
m2['fare'] = (m2['fare']-m2['fare'].min()) / (m2['fare'].max()-m2['fare'].min())
```

Out[354...]

	age_days	fare
--	----------	------

	age_days	fare
0	0.271174	0.014151
1	0.472229	0.139136
2	0.321438	0.015469
3	0.434531	0.103644
4	0.434531	0.015713

In [355...]

`m2.min()`

Out[355...]

```
age_days    0.0
fare        0.0
dtype: float64
```

In [356...]

`m2.max()`

Out[356...]

```
age_days    1.0
fare        1.0
dtype: float64
```

### Z-score Scaling Method ! (Standard Score)

- Range.(-3 to +3)

In [357...]

```
m3 = boat2[['age_days', 'fare']]
m3.head()
```

Out[357...]

	age_days	fare
0	8030.0	7.2500
1	13870.0	71.2833
2	9490.0	7.9250
3	12775.0	53.1000

	age_days	fare
4	12775.0	8.0500

In [358...]

```
m3['age_days'] = (m3['age_days']-m3['age_days'].mean()) / (m3['age_days'].std())
m3['fare'] = (m3['fare']-m3['fare'].mean()) / (m3['fare'].std())
m3.head()
```

C:\Users\fahad\AppData\Local\Temp\ipykernel\_11712/3802698713.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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
m3['age_days'] = (m3['age_days']-m3['age_days'].mean()) / (m3['age_days'].std())
C:\Users\fahad\AppData\Local\Temp\ipykernel_11712/3802698713.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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
m3['fare'] = (m3['fare']-m3['fare'].mean()) / (m3['fare'].std())
```

Out[358...]

	age_days	fare
0	-0.530005	-0.502163
1	0.571430	0.786404
2	-0.254646	-0.488580
3	0.364911	0.420494
4	0.364911	-0.486064

In [359...]

```
m3.min()
```

Out[359...]

```
age_days    -2.015566
fare        -0.648058
dtype: float64
```

In [360...]

```
m3.max()
```

```
Out[360... age_days    3.462699
          fare      9.661740
          dtype: float64
```

## log transformation

- Range.(1 to 1000)

```
In [361... m4 = boat2[['age_days', 'fare']]
m4.head()
```

	age_days	fare
0	8030.0	7.2500
1	13870.0	71.2833
2	9490.0	7.9250
3	12775.0	53.1000
4	12775.0	8.0500

```
In [362... m4['age_days'] = np.log(m4['age_days'])
m4['fare'] = np.log(m4['fare'])

m4.head()
```

C:\Users\fahad\AppData\Local\Temp\ipykernel\_11712\1973992703.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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
m4['age_days'] = np.log(m4['age_days'])
c:\python38\lib\site-packages\pandas\core\arraylike.py:364: RuntimeWarning: divide by zero encountered in log
    result = getattr(ufunc, method)(*inputs, **kwargs)
C:\Users\fahad\AppData\Local\Temp\ipykernel_11712\1973992703.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](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
iew-versus-a-copy  
m4['fare'] = np.log(m4['fare'])
```

```
Out[362...]
```

	age_days	fare
0	8.990940	1.981001
1	9.537484	4.266662
2	9.157994	2.070022
3	9.455245	3.972177
4	9.455245	2.085672

```
In [363...]
```

```
m4.min()
```

```
Out[363...]
```

```
age_days    5.032397  
fare        -inf  
dtype: float64
```

```
In [364...]
```

```
m4.max()
```

```
Out[364...]
```

```
age_days    10.281924  
fare        6.238967  
dtype: float64
```

## Binning

*Parametric Analysis tests*

*Non-Parametric Analysis tests*

```
In [365...]
```

```
boat4 = boat[['age', 'sex', 'who']]
```

```
In [366...]
```

```
boat4.age.head()
```

```
Out[366...]
```

1	38.0
3	35.0

```
6      54.0
10     4.0
11     58.0
Name: age, dtype: float64
```

In [367... boat4.age.max()

Out[367... 80.0

In [368... # bins = np.linspace(min(boat4['age']), max(boat4['age']), num=80)
# age\_groups = ['child', 'young', 'older']

# boat4['age'] = pd.cut(boat4['age'], bins, labels=age\_groups, include\_lowest=True)
# boat4['age']

## Converting Categorical variables into dummies !

In [369... boat4.sex.head()

Out[369... 1 female
3 female
6 male
10 female
11 female
Name: sex, dtype: object

In [370... boat4 = pd.get\_dummies(boat4, columns=[ 'sex'])

In [371... boat4.head()

	age	who	sex_female	sex_male
<b>1</b>	38.0	woman	1	0
<b>3</b>	35.0	woman	1	0
<b>6</b>	54.0	man	0	1
<b>10</b>	4.0	child	1	0

	age	who	sex_female	sex_male
11	58.0	woman	1	0

In [372...]: boat4.who.head()

Out[372...]:

```
1    woman
3    woman
6    man
10   child
11   woman
Name: who, dtype: object
```

In [373...]: boat4 = pd.get\_dummies(boat4, columns=['who'])

In [374...]: boat4.head()

Out[374...]:

	age	sex_female	sex_male	who_child	who_man	who_woman
1	38.0	1	0	0	0	1
3	35.0	1	0	0	0	1
6	54.0	0	1	0	1	0
10	4.0	1	0	1	0	0
11	58.0	1	0	0	0	1

# Machine Learning with Python | Crash Course

## (7). Machine Learning

### Simple\_Linear\_Regression

Dataset: \*\*Salary\_Prediction\*\*

#### Step: 01: (Import Dataset)

In [375...]

```
# import Dataset !
import pandas as pd

df = pd.read_csv("salary_data.csv")
df.head()
```

Out[375...]

	YearsExperience	Salary
0	1.1	39343
1	1.3	46205
2	1.5	37731
3	2.0	43525
4	2.2	39891

#### Step: 02: Data Already Cleaned or Organize

#### Step: 03: Splitting Dataset into (Training and Testing) Data

In [376...]

```
X = df[['YearsExperience']]
y = df[['Salary']]
```

In [377...]

```
X.head()
```

Out[377...]

	YearsExperience
0	1.1
1	1.3
2	1.5
3	2.0
4	2.2

In [378...]

```
y.head()
```

Out[378...]

	Salary
0	39343
1	46205
2	37731
3	43525
4	39891

In [379...]

```
# import library for splitting !
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

## Step: 04: Fit Linear-Regression Model

In [380...]

```
# import model !
from sklearn.linear_model import LinearRegression

model = LinearRegression().fit(X_train, y_train)
model
```

Out[380... LinearRegression()

## Step: 05: Plotting

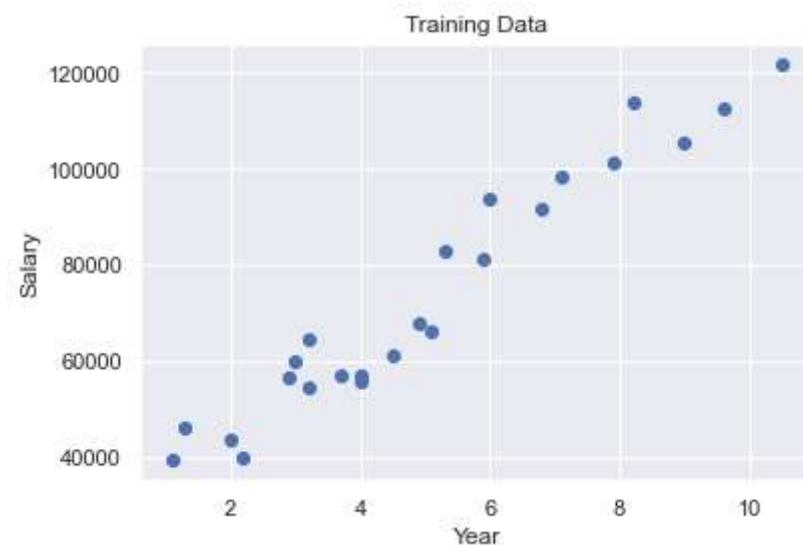
In [381...]

```
# import libraries !
import matplotlib.pyplot as plt
import seaborn as sns
```

In [382...]

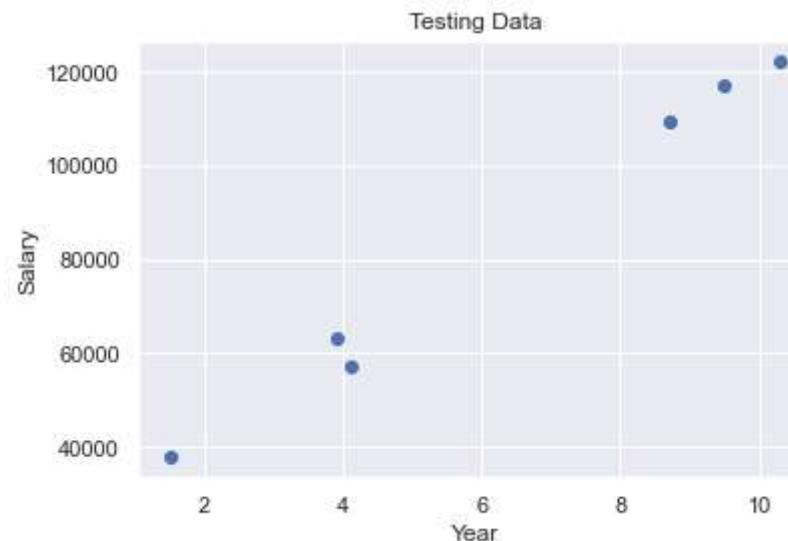
```
sns.set_theme(style="darkgrid", color_codes=True)
plt.scatter(X_train, y_train)
# plt.plot(X_train, model.predict(X_train), color="red")
plt.title('Training Data')
plt.xlabel('Year')
plt.ylabel('Salary')
```

Out[382...]



In [383...]

```
sns.set_theme(style="darkgrid", color_codes=True)
plt.scatter(X_test, y_test)
# plt.plot(X_test, model.predict(X_test), color="blue")
plt.title('Testing Data')
plt.xlabel('Year')
plt.ylabel('Salary')
```

Out[383...]  
Text(0, 0.5, 'Salary')

## Step: 06: Evaluate or Test Model Accuracy

In [384...]  

```
# Model fitness test !
print("Score of Training: ", model.score(X_train, y_train)*100)

print("Score of Testing: ", model.score(X_test, y_test)*100)
```

Score of Training: 94.11949620562126

Score of Testing: 98.8169515729126

## Step: 07: Prediction of Un-known values

In [385...]  

```
model.predict([[1]])
```

c:\python38\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names  
warnings.warn(  
array([[36092.67427736]]))

Out[385...]  
array([[36092.67427736]])In [386...]  

```
model.predict(X_test)
```

```
Out[386... array([[ 40748.96184072],  
 [122699.62295594],  
 [ 64961.65717022],  
 [ 63099.14214487],  
 [115249.56285456],  
 [107799.50275317]])
```

```
In [387... model.predict([[2], [4], [6], [8]])
```

```
c:\python38\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid feature names, but LinearRegression  
was fitted with feature names
```

```
    warnings.warn(
```

```
Out[387... array([[ 45405.24940409],  
 [ 64030.39965754],  
 [ 82655.549911 ],  
 [101280.70016446]])
```

```
In [388... f = [[10], [20], [30]]  
  
model.predict(f)
```

```
c:\python38\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid feature names, but LinearRegression  
was fitted with feature names
```

```
    warnings.warn(
```

```
Out[388... array([[119905.85041792],  
 [213031.60168521],  
 [306157.3529525 ]])
```

## Multiple\_Linear\_Regression

### \*\*Dataset: Salary\_Prediction\*\*

#### Step: 01: (Import Dataset)

```
In [389... # import Dataset !  
import pandas as pd
```

```
df = pd.read_csv("ml_data_salary.csv")
df.head()
```

Out[389...]

	age	distance	YearsExperience	Salary
<b>0</b>	31.1	77.75	1.1	39343
<b>1</b>	31.3	78.25	1.3	46205
<b>2</b>	31.5	78.75	1.5	37731
<b>3</b>	32.0	80.00	2.0	43525
<b>4</b>	32.2	80.50	2.2	39891

## Step: 02: Data Normalization

In [390...]

```
features = df[['age', 'distance', 'YearsExperience']]
features.head()
```

Out[390...]

	age	distance	YearsExperience
<b>0</b>	31.1	77.75	1.1
<b>1</b>	31.3	78.25	1.3
<b>2</b>	31.5	78.75	1.5
<b>3</b>	32.0	80.00	2.0
<b>4</b>	32.2	80.50	2.2

In [391...]

```
# features['age'] = features['age']/features['age'].max()
# features['distance'] = features['distance']/features['distance'].max()
# features['YearsExperience'] = features['YearsExperience']/features['YearsExperience'].max()

# features.head()
```

In [392...]

```
# # Data transformation
# from sklearn.preprocessing import MinMaxScaler

# scaler = MinMaxScaler()
```

```
# features = scaler.fit_transform(features)
# features
```

## Step: 03: Splitting Dataset into (Training and Testing) Data

In [393...]

```
X = features
y = df[['Salary']]
```

In [394...]

```
X.head()
```

Out[394...]

	age	distance	YearsExperience
0	31.1	77.75	1.1
1	31.3	78.25	1.3
2	31.5	78.75	1.5
3	32.0	80.00	2.0
4	32.2	80.50	2.2

In [395...]

```
y.head()
```

Out[395...]

	Salary
0	39343
1	46205
2	37731
3	43525
4	39891

In [396...]

```
# import library for splitting !
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

## Step: 04: Fit Linear-Regression Model

In [397...]

```
# import model !
from sklearn.linear_model import LinearRegression

model = LinearRegression().fit(X_train, y_train)
model
```

Out[397...]

```
LinearRegression()
```

In [398...]

```
model.coef_
```

Out[398...]

```
array([[ 1.53152206e+16,  1.89735921e+14, -1.57895604e+16]])
```

In [399...]

```
model.intercept_
```

Out[399...]

```
array([-4.73686812e+17])
```

## Step: 05: Plotting

In [400...]

```
# import libraries !
import matplotlib.pyplot as plt
import seaborn as sns
```

In [401...]

```
# sns.set_theme(style="whitegrid", color_codes=True)
# plt.scatter(X_train, y_train)
# plt.plot(X_train, model.predict(X_train), color="red")
# plt.title('Training Data')
# plt.xlabel('Year')
# plt.ylabel('Salary')
```

## Step: 06: Evaluate or Test Model Accuracy

In [402...]

```
# model.score(X_train, y_train)
model.score(X_train, y_train)*100
```

Out[402...]  
94.10812914659125

## Step: 07: Prediction of Un-known values

In [403...]  

```
model.predict([[31.1, 77.75, 1.1]])
```

```
c:\python38\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid feature names, but LinearRegression  
was fitted with feature names  
    warnings.warn(  
Out[403...]  
array([[36992.]])
```

---

## Decision\_Tree\_Classifier

### \*\*Dataset: Food\_Prediction\*\*

#### Step: 01: (Import Dataset)

In [404...]  

```
# import Dataset !  
import pandas as pd  
  
df = pd.read_csv("chillamldata.csv")  
df.head()
```

Out[404...]  

	age	height	weight	gender	likeness
0	27	170.688	76.0	Male	Biryani
1	41	165.000	70.0	Male	Biryani
2	29	171.000	80.0	Male	Biryani
3	27	173.000	102.0	Male	Biryani
4	29	164.000	67.0	Male	Biryani

#### Step: 02: Data be Normalized !</span>

```
In [405... df['gender'] = df['gender'].replace('Male', 1)
df['gender'] = df['gender'].replace('Female', 0)
```

```
In [406... # selection of input & output variables !
X = df.drop(['likeness'], axis=1)
y = df[['likeness']]
```

```
In [407... X.head()
```

```
Out[407...   age  height  weight  gender
 0    27  170.688     76.0      1
 1    41  165.000     70.0      1
 2    29  171.000     80.0      1
 3    27  173.000    102.0      1
 4    29  164.000     67.0      1
```

```
In [408... y.head()
```

```
Out[408...   likeness
 0    Biryani
 1    Biryani
 2    Biryani
 3    Biryani
 4    Biryani
```

## Step: 03: Splitting Dataset into (Training and Testing) Data

```
In [409... # import library for splitting !
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

## Step: 04: Fit Decision Tree Model

In [410...]

```
# import model !
from sklearn.tree import DecisionTreeClassifier

# Create & Fit Model
model = DecisionTreeClassifier().fit(X_train, y_train)
model
```

Out[410...]

```
DecisionTreeClassifier()
```

## Step: 05: Evaluate or Test Model Accuracy

In [411...]

```
predicted_values = model.predict(X_test)
predicted_values

# import library for score !
from sklearn.metrics import accuracy_score

score = accuracy_score(y_test, predicted_values)*100
score
```

Out[411...]

```
44.89795918367347
```

## Step: 06: Prediction of Un-known values

In [412...]

```
model.predict([[27, 170.688, 76.0, 1]])
```

```
c:\python38\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
```

```
warnings.warn(
```

Out[412...]

```
array(['Biryani'], dtype=object)
```

In [413...]

```
model.predict([[22, 163.688, 63.0, 1]])
```

```
c:\python38\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid feature names, but DecisionTreeClassifier was fitted with feature names
  warnings.warn(
array(['Samosa'], dtype=object)
Out[413...]
```

## How to save model !

```
In [414...]
# import libraries !
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
import joblib as jb

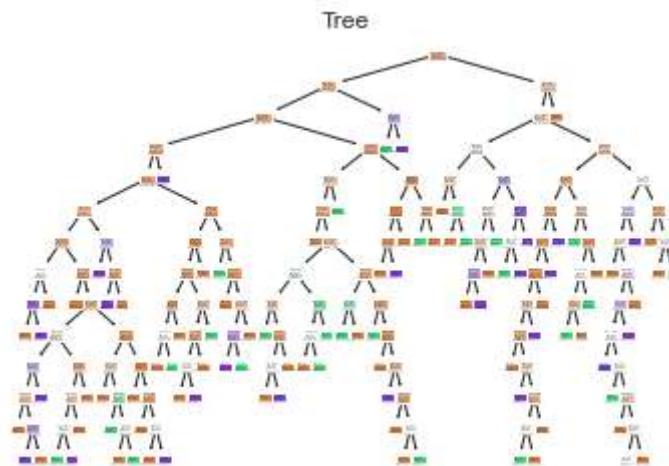
model = DecisionTreeClassifier()
jb.dump(model, 'foodie.joblib')

Out[414...]
['foodie.joblib']
```

## Graph

```
In [415...]
# graph
from sklearn.tree import plot_tree
from sklearn.tree import DecisionTreeClassifier
import matplotlib.pyplot as plt

model1 = DecisionTreeClassifier().fit(X,y)
plot_tree(model1, filled=True)
plt.title("Tree")
plt.show()
```



---

## Decision\_Tree\_Classifier

**\*\*Dataset: iris\*\***

### Task

In [416...]

```
import pandas as pd  
import seaborn as sns
```

In [417...]

```
df = sns.load_dataset('iris')  
df.head()
```

Out[417...]

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
In [418... df.isnull().sum()
```

```
Out[418... sepal_length    0  
sepal_width     0  
petal_length    0  
petal_width     0  
species         0  
dtype: int64
```

```
In [419... # splitting data !  
X = df.drop(['species'], axis=1)  
y = df[['species']]
```

```
In [420... X.head(2)
```

```
Out[420...  
sepal_length  sepal_width  petal_length  petal_width  
0            5.1          3.5          1.4          0.2  
1            4.9          3.0          1.4          0.2
```

```
In [421... y.head(2)
```

```
Out[421...  
species  
0  setosa  
1  setosa
```

```
In [422... # train / test data !  
from sklearn.model_selection import train_test_split  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size= 0.2, random_state= 1)
```

```
In [423... # create & fit model !  
from sklearn.tree import DecisionTreeClassifier  
  
model = DecisionTreeClassifier()
```

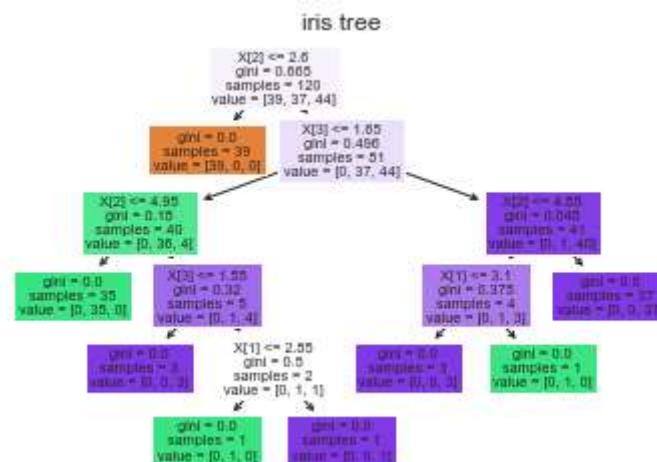
```
model.fit(X_train, y_train)
model
```

Out[423... DecisionTreeClassifier()

In [424...

```
# Draw graph !
from sklearn.tree import plot_tree
import matplotlib.pyplot as plt
import seaborn as sns

sns.set_theme(style="white", color_codes=True)
plot_tree(model, filled=True)
plt.title('iris tree')
plt.show()
```



In [425...

```
pred_val = model.predict(X_test)
pred_val

# r^2 method !
from sklearn.metrics import accuracy_score
r2_score = accuracy_score(y_test, pred_val)*100
r2_score
```

Out[425... 96.66666666666667

In [426...

```
# check Accuracy !
model.score(X_train, y_train)*100
```

Out[426... 100.0

In [427... # check Accuracy !
model.score(X, y)\*100

Out[427... 99.33333333333333

## Random-Forest\_Classifier

**\*\*Dataset: iris\*\***

### Task

In [428...  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt

In [429...  
flowers = sns.load\_dataset('iris')  
flowers.head()

Out[429...  

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [430...]

```
X = flowers.drop(['species'], axis=1)
y = flowers[['species']]
```

In [431...]

```
X.head(1)
```

Out[431...]

	sepal_length	sepal_width	petal_length	petal_width
<b>0</b>	5.1	3.5	1.4	0.2

In [432...]

```
y.head(1)
```

Out[432...]

	species
<b>0</b>	setosa

In [433...]

```
# splitting !
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=1)
```

standard (no's of trees)n\_estimators is (10 to 100) ... (like, RandomForestClassifier(n\_estimators=(10 to 100)))

In [434...]

```
from sklearn.ensemble import RandomForestClassifier
```

In [435...]

```
model1 = RandomForestClassifier(n_estimators=10).fit(X, y)
model1.predict([[10, 20, 30, 40]])
```

C:\Users\fahad\AppData\Local\Temp\ipykernel\_11712\2813647156.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n\_samples,), for example using ravel().

```
model1 = RandomForestClassifier(n_estimators=10).fit(X, y)
c:\python38\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid feature names, but RandomForestClassifier was fitted with feature names
    warnings.warn(
        array(['virginica'], dtype=object)
```

Out[435...]

In [436...]

```
model = RandomForestClassifier().fit(X_train, y_train)
```

```
model
```

```
C:\Users\fahad\AppData\Local\Temp/ipykernel_11712/2183830046.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
```

```
    model = RandomForestClassifier().fit(X_train, y_train)
```

```
RandomForestClassifier()
```

```
Out[436...]
```

```
pred_value = model.predict(X_test)  
pred_value
```

```
Out[437...]
```

```
array(['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica',  
       'versicolor', 'virginica', 'setosa', 'setosa', 'virginica',  
       'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',  
       'setosa', 'versicolor', 'versicolor', 'setosa', 'setosa',  
       'versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',  
       'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica'],  
      dtype=object)
```

```
In [438...]
```

```
from sklearn.metrics import accuracy_score  
  
r2_score = accuracy_score(y_test, pred_value)*100  
r2_score
```

```
Out[438...]
```

```
96.66666666666667
```

```
In [439...]
```

```
# make confution metrics !  
from sklearn.metrics import confusion_matrix  
  
cm = confusion_matrix(y_test, pred_value)  
cm
```

```
Out[439...]
```

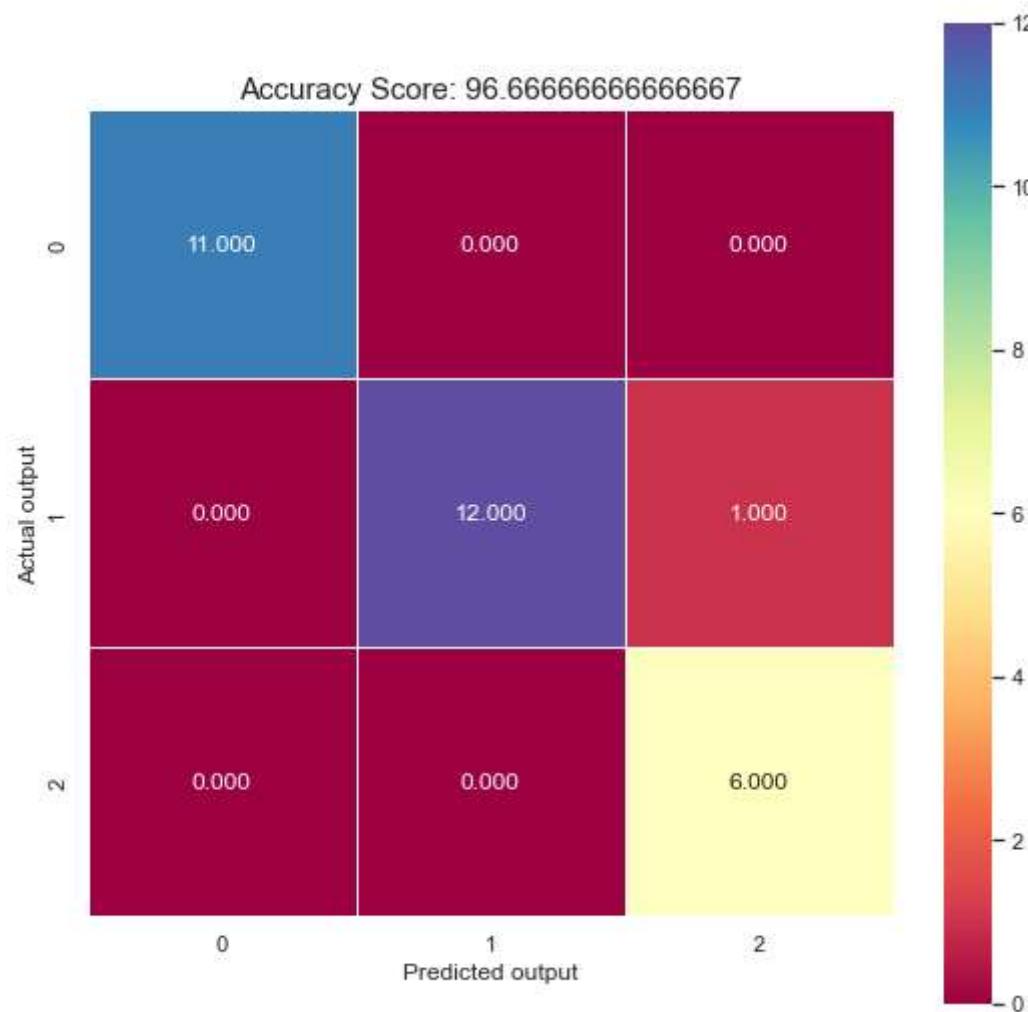
```
array([[11,  0,  0],  
       [ 0, 12,  1],  
       [ 0,  0,  6]], dtype=int64)
```

```
In [440...]
```

```
sns.set_theme(style="white", color_codes=True)  
plt.figure(figsize=(9,9))  
  
sns.heatmap(cm, annot= True, fmt= '.3f', linewidths= .5, square= True, cmap= 'Spectral')  
  
plt.xlabel("Predicted output")
```

```
plt.ylabel("Actual output")  
  
all_sample_title= "Accuracy Score: {0}".format(r2_score)  
  
plt.title(all_sample_title, size= '15')
```

Out[440... Text(0.5, 1.0, 'Accuracy Score: 96.66666666666667')



## K-Nearest Neighbors

## \*\*Dataset: Chilla Dataset !\*\*

### Step: 01: (Import Dataset)

In [441...]

```
# import Dataset !
import pandas as pd

df = pd.read_csv('chillamldata.csv')
df.head()
```

Out[441...]

	age	height	weight	gender	likeness
0	27	170.688	76.0	Male	Biryani
1	41	165.000	70.0	Male	Biryani
2	29	171.000	80.0	Male	Biryani
3	27	173.000	102.0	Male	Biryani
4	29	164.000	67.0	Male	Biryani

In [442...]

```
df['gender'] = df['gender'].replace('Male', 1)
df['gender'] = df['gender'].replace('Female', 0)
```

In [443...]

```
X = df.drop('likeness', axis=1)
y = df.likeness
```

In [444...]

```
X.head(2)
```

Out[444...]

	age	height	weight	gender
0	27	170.688	76.0	1
1	41	165.000	70.0	1

In [445...]

```
y.head(2)
```

```
Out[445... 0    Biryani  
1    Biryani  
Name: likeness, dtype: object
```

## Step: 02: Data Cleaned or Organize

```
In [446... df.isnull().sum()
```

```
Out[446... age      0  
height    0  
weight    0  
gender    0  
likeness  0  
dtype: int64
```

## Step: 03: Splitting Dataset into (Training and Testing) Data

```
In [447... from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

## Step: 04: Fit K-Nearest\_Neighbor Model

```
In [448... from sklearn.neighbors import KNeighborsClassifier  
  
model = KNeighborsClassifier(n_neighbors=5).fit(X_train, y_train)  
model
```

```
Out[448... KNeighborsClassifier()
```

## Step: 05: Evaluate or Test Model Accuracy

```
In [449... predicted_values = model.predict(X_test)  
predicted_values  
  
from sklearn.metrics import accuracy_score  
  
score = accuracy_score(y_test, predicted_values)*100  
print("K-NN. (r2.score):", score)
```

K-NN. (r2.score): 57.14285714285714

## Step: 06: Prediction of Un-known values

In [450...]: model.predict(X\_test)

```
Out[450]: array(['Biryani', 'Samosa', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
   'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Samosa', 'Biryani',
   'Samosa', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
   'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
   'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
   'Biryani', 'Biryani', 'Biryani', 'Samosa', 'Biryani', 'Samosa',
   'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Samosa',
   'Samosa', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Samosa',
   'Pakora', 'Biryani', 'Biryani', 'Biryani', 'Biryani', 'Biryani',
   'Biryani'], dtype=object)
```

# Polynomial Regression

## \*\*Dataset: Salaries !\*\*

```
In [451...]: # import libraries!
```

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

```
In [452]: dataset = pd.read_csv('Position_Salaries.csv')  
dataset.head(3)
```

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000

```
In [453... X = dataset[['Level']]  
y = dataset[['Salary']]
```

```
In [454... X.head(2)
```

```
Out[454... Level
```

	Level
0	1
1	2

```
In [455... X.head()
```

```
Out[455... Level
```

	Level
0	1
1	2
2	3
3	4
4	5

```
In [456... y.head(2)
```

```
Out[456... Salary
```

	Salary
0	45000
1	50000

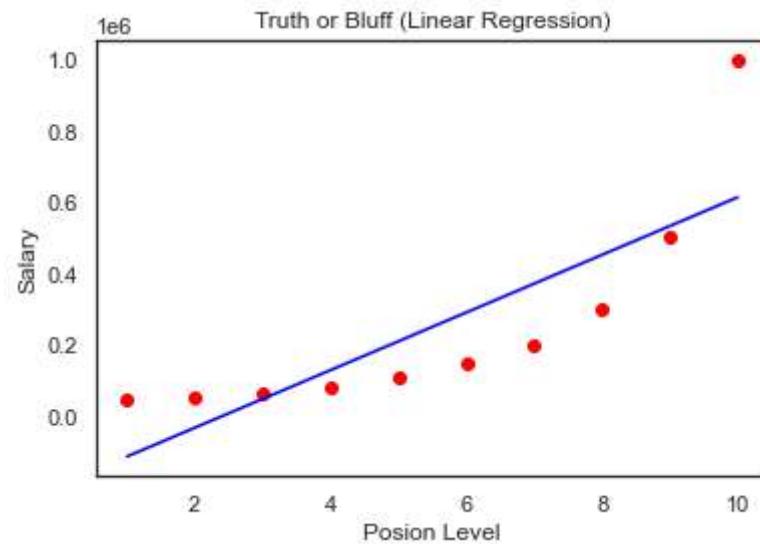
```
In [457... # Splitting  
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

```
In [458...
```

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X, y)

sns.set_theme(style="white", color_codes=True)
plt.scatter(X, y, color='red')
plt.plot(X, model.predict(X), color='blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
plt.ylabel('Salary')
plt.show()
```



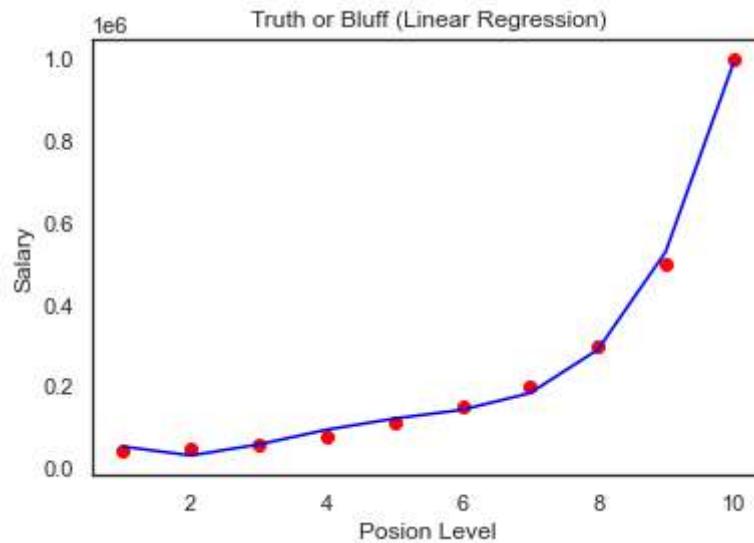
In [459...]

```
# fit polynomial regression to dataset
from sklearn.preprocessing import PolynomialFeatures

poly_reg = PolynomialFeatures(degree=4)
X_poly = poly_reg.fit_transform(X)
poly_reg = LinearRegression()
poly_reg.fit(X_poly, y)

sns.set_theme(style="white", color_codes=True)
plt.scatter(X, y, color='red')
plt.plot(X, poly_reg.predict(X_poly), color='blue')
plt.title('Truth or Bluff (Linear Regression)')
plt.xlabel('Position Level')
```

```
plt.ylabel('Salary')
plt.show()
```



In [460...]

```
pred_linear = model.predict([[11]])
pred_linear
```

c:\python38\lib\site-packages\sklearn\base.py:445: UserWarning: X does not have valid feature names, but LinearRegression was fitted with feature names  
warnings.warn(  
array([694333.333333333])]

Out[460...]

## Naive Bayes

**\*\*Dataset: iris !\*\***

In [461...]

```
# import Libraries !

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

## Step: 01: (Import Dataset)

In [462...]

```
df = sns.load_dataset('iris')
df.head()
```

Out[462...]

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [463...]

```
X = df.drop('species', axis=1)
y = df[['species']]
```

In [464...]

```
X.head(2)
```

Out[464...]

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

In [465...]

```
y.head(2)
```

Out[465...]

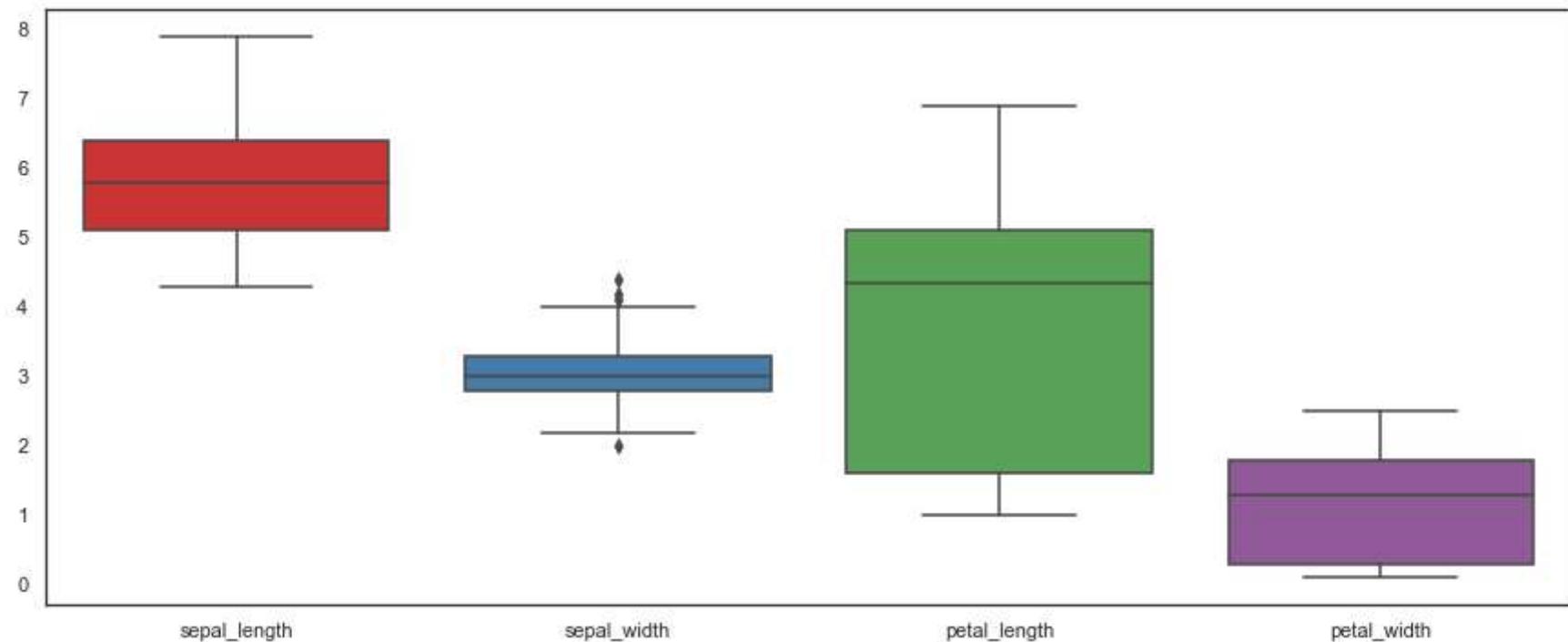
	species
0	setosa
1	setosa

## Step: 02: Data Cleaned or Organize

```
In [466]: df.isnull().sum()
```

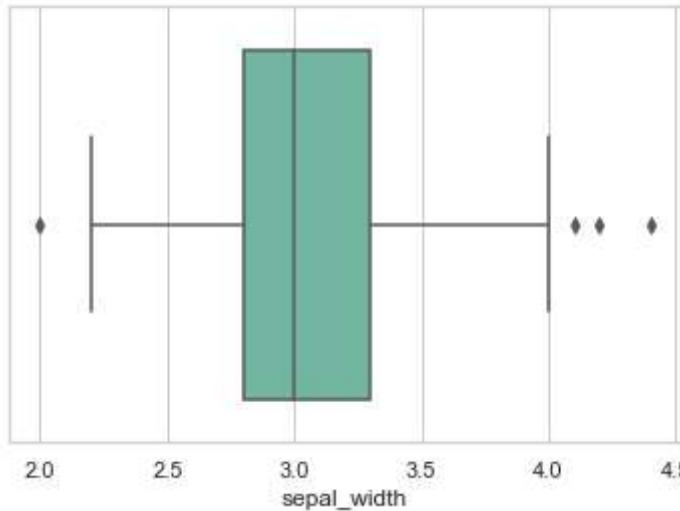
```
Out[466]: sepal_length    0  
          sepal_width     0  
          petal_length    0  
          petal_width     0  
          species        0  
          dtype: int64
```

```
In [467]: plt.figure(figsize=(15,6))  
sns.set_theme(style="white", color_codes=True)  
sns.boxplot(data=df,  
            palette="Set1")  
plt.show()
```



```
In [468]: sns.set_theme(style="whitegrid", color_codes=True)  
sns.boxplot(data=df,  
            x="sepal_width",
```

```
palette="Set2")
plt.show()
```

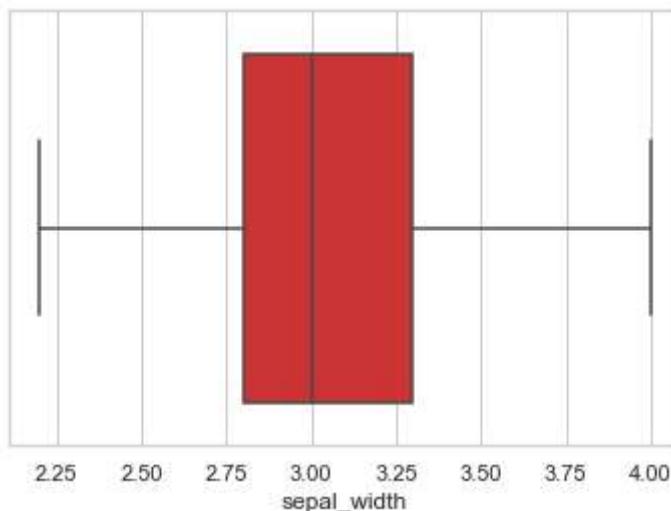


In [469...]

```
df = df[df['sepal_width']<4.1]
df = df[df['sepal_width']>2.1]
```

In [470...]

```
sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
             x="sepal_width",
             palette="Set1")
plt.show()
```



## Step: 03: Splitting Dataset into (Training and Testing) Data

In [471...]

```
from sklearn.model_selection import train_test_split  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=1)
```

## Step: 04: Fit Naive Bayes Model

In [472...]

```
from sklearn.naive_bayes import GaussianNB  
  
model = GaussianNB().fit(X_train, y_train)  
model
```

c:\python38\lib\site-packages\sklearn\utils\validation.py:985: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n\_samples, ), for example using ravel().

```
y = column_or_1d(y, warn=True)
```

Out[472...]

## Step: 05: Evaluate or Test Model Accuracy

In [473...]

```
predicted_values = model.predict(X_test)  
predicted_values
```

```
from sklearn.metrics import accuracy_score

score = round(accuracy_score(y_test, predicted_values)*100, 4)
print("GaussianNB. (r2.score):", score)
```

GaussianNB. (r2.score): 97.3684

## Step: 06: Prediction of Un-known values

In [474...]: model.predict(X\_test)

Out[474...]: array(['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica', 'versicolor', 'virginica', 'setosa', 'setosa', 'virginica', 'versicolor', 'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor', 'setosa', 'versicolor', 'setosa', 'setosa', 'versicolor', 'versicolor', 'setosa', 'virginica', 'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica', 'versicolor', 'virginica', 'versicolor', 'versicolor', 'virginica', 'versicolor', 'virginica', 'setosa', 'versicolor', 'setosa'], dtype='|<U10')

## Support\_Vector\_Machine

### \*\*Dataset: Breast Cancer !\*\*

In [475...]: # import dataset !
from sklearn import datasets

In [476...]: # dataset
cancer = datasets.load\_breast\_cancer()

In [477...]: cancer.data.shape

Out[477...]: (569, 30)

In [478...]: # print the names of features !. (Independent)

```
print("Features: ", cancer.feature_names)

# print the Label type of cancer ! (Dependent)
print("Targets: ", cancer.target_names)
```

```
Features: ['mean radius' 'mean texture' 'mean perimeter' 'mean area'  
          'mean smoothness' 'mean compactness' 'mean concavity'  
          'mean concave points' 'mean symmetry' 'mean fractal dimension'  
          'radius error' 'texture error' 'perimeter error' 'area error'  
          'smoothness error' 'compactness error' 'concavity error'  
          'concave points error' 'symmetry error' 'fractal dimension error'  
          'worst radius' 'worst texture' 'worst perimeter' 'worst area'  
          'worst smoothness' 'worst compactness' 'worst concavity'  
          'worst concave points' 'worst symmetry' 'worst fractal dimension']  
Targets: ['malignant' 'benign']
```

In [479]:

```
features = cancer.data
```

Out[479]

```
array([[1.799e+01, 1.038e+01, 1.228e+02, ... , 2.654e-01, 4.601e-01,
       1.189e-01],
       [2.057e+01, 1.777e+01, 1.329e+02, ... , 1.860e-01, 2.750e-01,
        8.902e-02],
       [1.969e+01, 2.125e+01, 1.300e+02, ... , 2.430e-01, 3.613e-01,
        8.758e-02],
       ... ,
       [1.660e+01, 2.808e+01, 1.083e+02, ... , 1.418e-01, 2.218e-01,
        7.820e-02],
       [2.060e+01, 2.933e+01, 1.401e+02, ... , 2.650e-01, 4.087e-01,
        1.240e-01],
       [7.760e+00, 2.454e+01, 4.792e+01, ... , 0.000e+00, 2.871e-01,
        7.039e-02]])
```

In [480...]

```
target = cancer.target
```

Out[480]

```
1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
1, 1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0,
0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0,
```

In [481...]

```
# data splitting !
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=0)
```

In [482...]

```
# model !
from sklearn import svm

clf_svm = svm.SVC(kernel='linear').fit(X_train, y_train) # Linear kernel
clf_svm
```

Out[482...]

SVC(kernel='linear')

In [483...]

```
y_pred = clf_svm.predict(X_test)
y_pred
```

Out[483...]

```
array([0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1,
       0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1,
       0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1,
       0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 1])
```

In [484...]

```
# accuracy !
from sklearn import metrics

r2_score = round(metrics.accuracy_score(y_test, y_pred)*100, 4)
print("accuracy:", r2_score)
```

accuracy: 95.614

In [485...]

```
# precision !
from sklearn import metrics

precision = round(metrics.precision_score(y_test, y_pred)*100, 4)
print("r2_score:", precision)
```

r2\_score: 98.4375

In [486...]

```
# precision !
from sklearn import metrics

recall = round(metrics.recall_score(y_test, y_pred)*100, 4)
print("r2_score:", recall)
```

r2\_score: 94.0299

In [487...]

```
# confusion matrices !
from sklearn import metrics

cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

Out[487...]

```
array([[46,  1],
       [ 4, 63]], dtype=int64)
```

## Clustering

### K-Mean Clustering

In [488...]

```
# import libraries !
```

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

In [489...]

```
# import dataset !

df = pd.read_csv('dataset_FreshCo.csv')
df.head()
```

Out[489...]

	homeshop	storeshop	employ	income	location	size	age
0	20	310	9	NaN	2	1	72
1	92	230	9	30550.0	2	2	60
2	69	240	10	64971.0	2	4	19
3	88	192	9	37298.0	4	2	63
4	49	42	1	41135.0	4	6	47

In [490...]

```
df.shape
```

Out[490...]

```
(2093, 7)
```

In [491...]

```
df.isnull().sum()
```

Out[491...]

```
homeshop      0
storeshop     0
employ        0
income       681
location      0
size          0
age           0
dtype: int64
```

In [492...]

```
# check empty (column %) !

for feature in df:
```

```
if df[feature].isnull().sum()>1:
    print(feature,":", np.round(df[feature].isnull().mean(),4)*100, '%')
```

income : 32.54 %

In [493...]: df['income'] = df['income'].replace(np.nan, df.income.mean())

In [494...]: df.income.isnull().sum()

Out[494...]: 0

In [495...]: df.head()

	homeshop	storeshop	employ	income	location	size	age
0	20	310	9	28871.518414	2	1	72
1	92	230	9	30550.000000	2	2	60
2	69	240	10	64971.000000	2	4	19
3	88	192	9	37298.000000	4	2	63
4	49	42	1	41135.000000	4	6	47

In [496...]: # scaling !

```
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler()

# feature = [i for i in df]
feature = df.columns
df[feature] = scale.fit_transform(df[feature])
df.head()
```

	homeshop	storeshop	employ	income	location	size	age
0	0.075000	0.498361	0.666667	0.321213	0.333333	0.000	0.811594
1	0.375000	0.367213	0.666667	0.339892	0.333333	0.125	0.637681

	homeshop	storeshop	employ	income	location	size	age
2	0.279167	0.383607	0.750000	0.722952	0.333333	0.375	0.043478
3	0.358333	0.304918	0.666667	0.414988	1.000000	0.125	0.681159
4	0.195833	0.059016	0.000000	0.457689	1.000000	0.625	0.449275

In [497...]

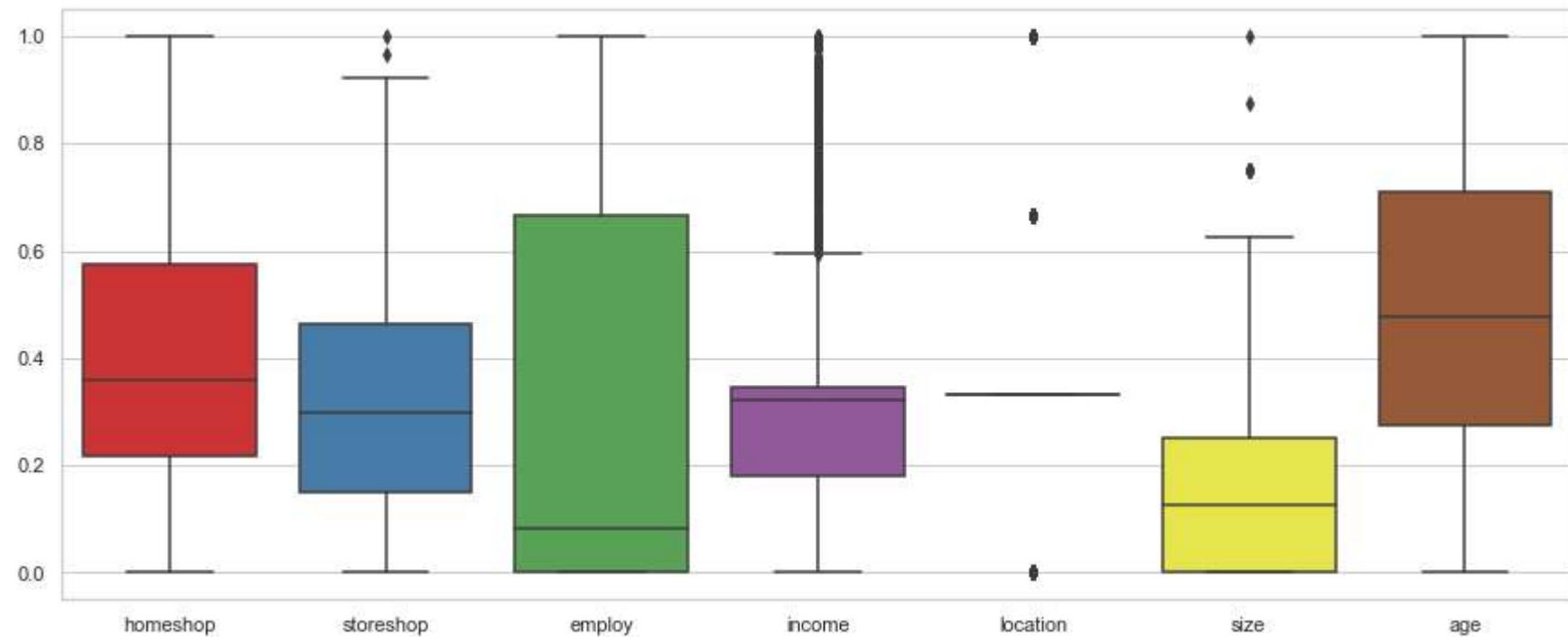
```
df.shape
```

Out[497...]

```
(2093, 7)
```

In [498...]

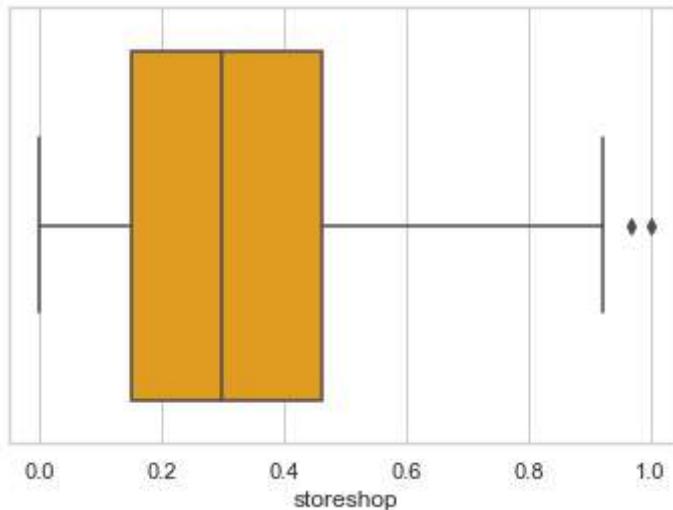
```
plt.figure(figsize=(15,6))
sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
            palette="Set1")
plt.show()
```



In [499...]

```
# storeshop

sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
            x="storeshop",
            color="orange")
plt.show()
```



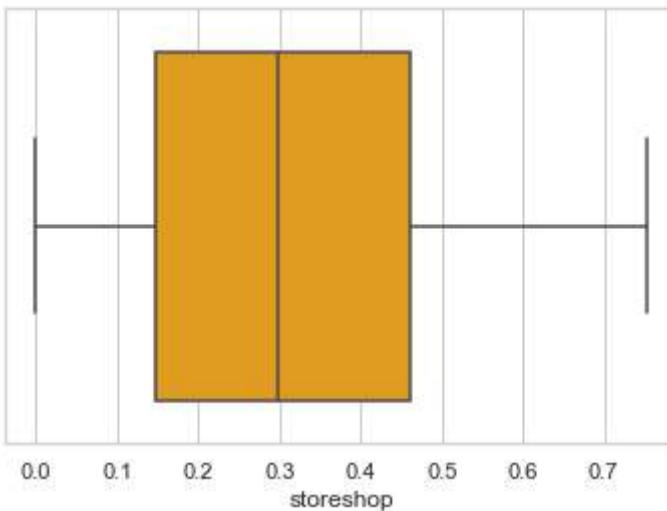
In [500...]

```
df = df[df['storeshop'] < 0.92]
```

In [501...]

```
# storeshop

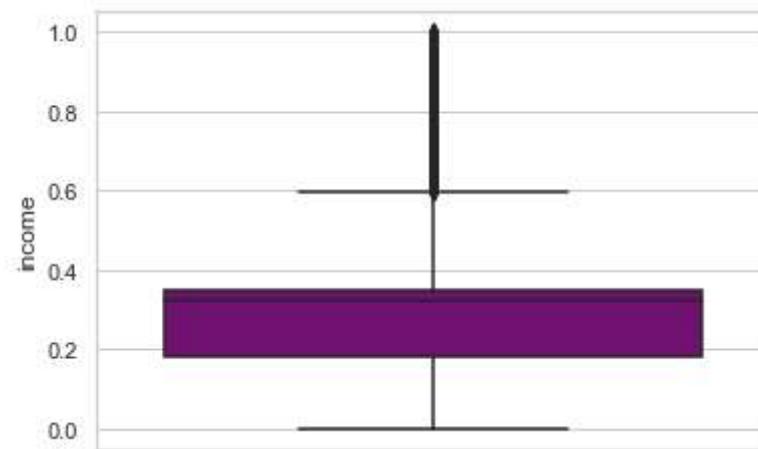
sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
            x="storeshop",
            color="orange")
plt.show()
```



In [502...]

```
# income

sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
            y="income",
            color="purple")
plt.show()
```



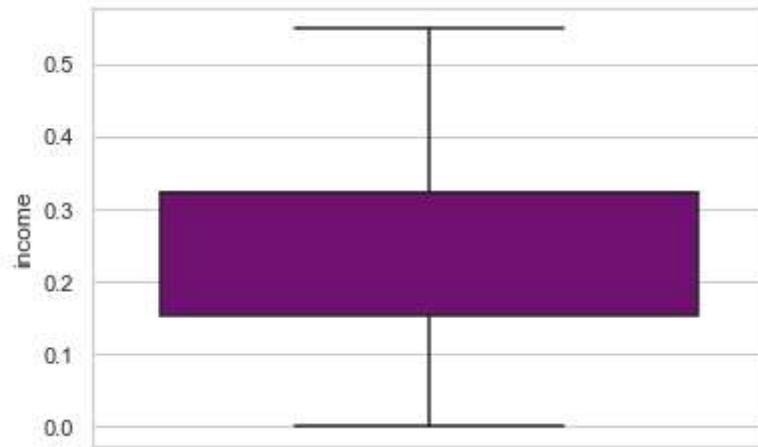
In [503...]

```
df = df[df['income'] < 0.55]
```

In [504...]

```
# income

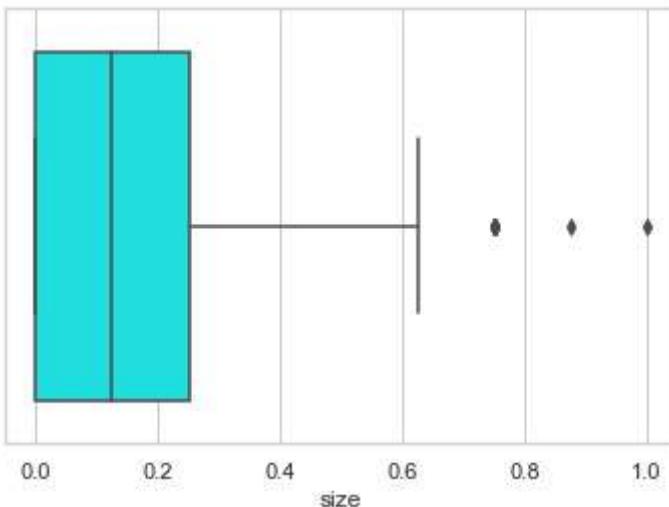
sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
            y="income",
            color="purple")
plt.show()
```



In [505...]

```
# size

sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
            x="size",
            color="cyan")
plt.show()
```



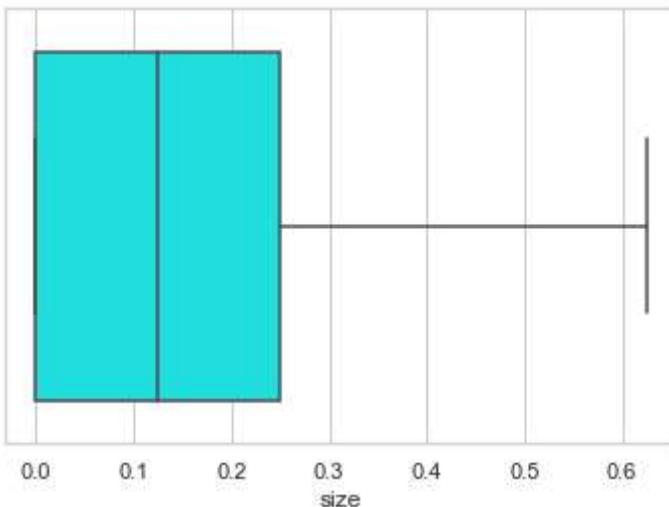
In [506...]

```
df = df[df['size']<0.7]
```

In [507...]

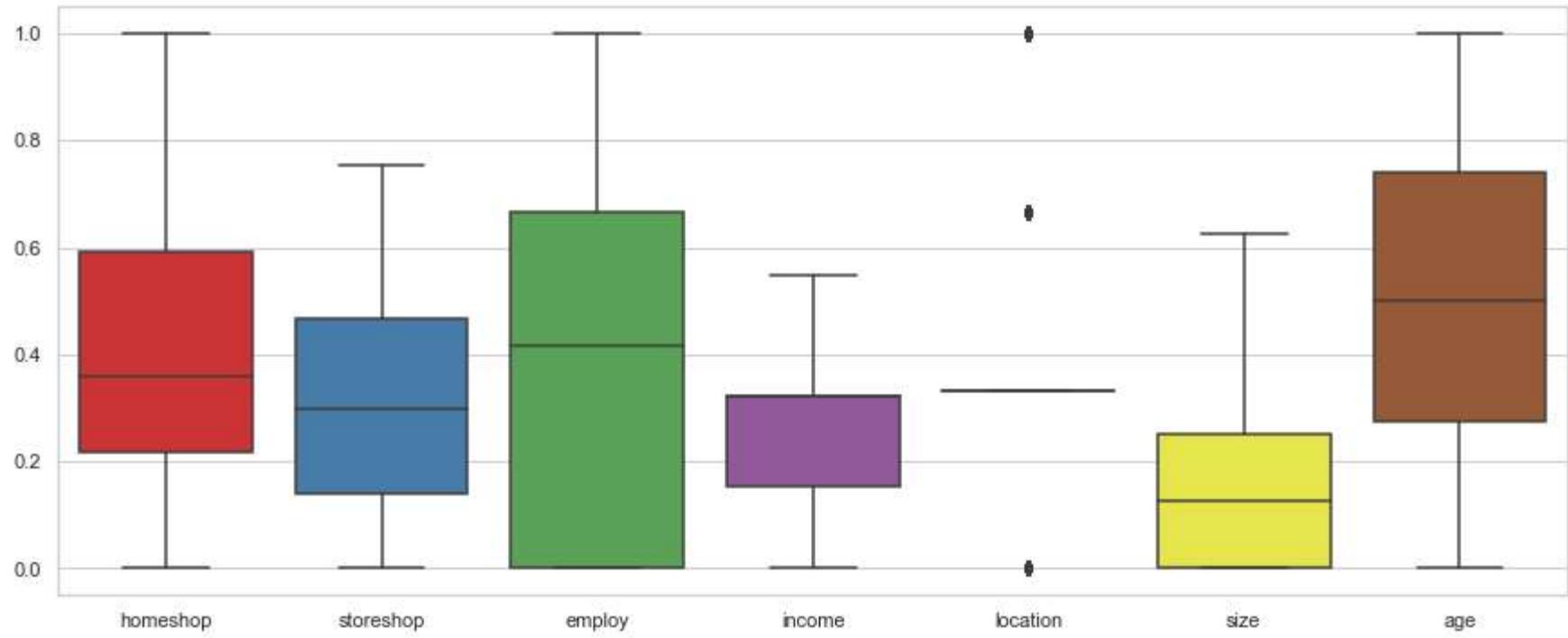
```
# size

sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
            x="size",
            color="cyan")
plt.show()
```



In [508...]

```
plt.figure(figsize=(15,6))
sns.set_theme(style="whitegrid", color_codes=True)
sns.boxplot(data=df,
            palette="Set1")
plt.show()
```



In [509...]: df.shape

Out[509...]: (1802, 7)

In [510...]: df.corr()

	homeshop	storeshop	employ	income	location	size	age
homeshop	1.000000	0.035334	0.024863	-0.028314	-0.031369	-0.085414	0.024872
storeshop	0.035334	1.000000	0.066341	-0.029724	-0.000752	-0.028723	0.089951
employ	0.024863	0.066341	1.000000	-0.235544	0.040185	-0.241091	0.522935
income	-0.028314	-0.029724	-0.235544	1.000000	0.028272	0.206271	-0.111635
location	-0.031369	-0.000752	0.040185	0.028272	1.000000	-0.063996	0.186306
size	-0.085414	-0.028723	-0.241091	0.206271	-0.063996	1.000000	-0.427604
age	0.024872	0.089951	0.522935	-0.111635	0.186306	-0.427604	1.000000

In [511...]

```
df.corr('spearman')
```

Out[511...]

	homeshop	storeshop	employ	income	location	size	age
homeshop	1.000000	0.042739	0.020479	-0.029298	-0.026507	-0.101423	0.029608
storeshop	0.042739	1.000000	0.060644	-0.029751	-0.009795	-0.026722	0.088454
employ	0.020479	0.060644	1.000000	-0.202976	0.051019	-0.231175	0.529576
income	-0.029298	-0.029751	-0.202976	1.000000	0.007903	0.224505	-0.109254
location	-0.026507	-0.009795	0.051019	0.007903	1.000000	-0.067336	0.188844
size	-0.101423	-0.026722	-0.231175	0.224505	-0.067336	1.000000	-0.421772
age	0.029608	0.088454	0.529576	-0.109254	0.188844	-0.421772	1.000000

In [512...]

```
pearson = df.corr('pearson')
```

In [513...]

```
# Correlation !  
  
df = pd.DataFrame(pearson)  
  
plt.figure(figsize=(12,6))  
corrMatrix = df.corr()  
sns.heatmap(corrMatrix, annot=True)  
plt.show()
```



In [514...]

```
# histogram

df.hist()
plt.show()
```



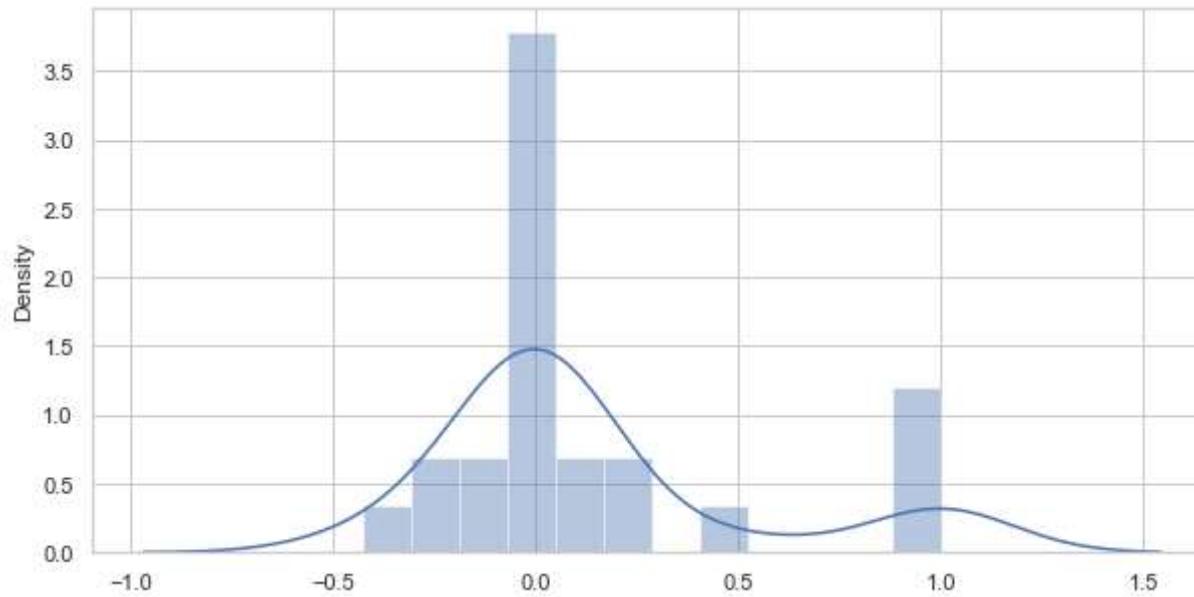
In [515...]

```
# displot (check: bell curve)

plt.figure(figsize=(10,5))
sns.set_theme(style="whitegrid", color_codes=True)
sns.distplot(df)
plt.show()
```

c:\python38\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)
```



In [516...]

```
from sklearn.cluster import KMeans  
  
k_mean = KMeans(n_clusters=3)  
k_mean
```

Out[516...]

```
KMeans(n_clusters=3)
```

In [517...]

```
y_predicted = k_mean.fit_predict(df)  
y_predicted
```

Out[517...]

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

In [518...]

```
df["Cluster"] = y_predicted
```

In [519...]

```
df.head()
```

Out[519...]

	homeshop	storeshop	employ	income	location	size	age	Cluster
homeshop	1.000000	0.035334	0.024863	-0.028314	-0.031369	-0.085414	0.024872	2

	homeshop	storeshop	employ	income	location	size	age	Cluster
storeshop	0.035334	1.000000	0.066341	-0.029724	-0.000752	-0.028723	0.089951	2
employ	0.024863	0.066341	1.000000	-0.235544	0.040185	-0.241091	0.522935	1
income	-0.028314	-0.029724	-0.235544	1.000000	0.028272	0.206271	-0.111635	0
location	-0.031369	-0.000752	0.040185	0.028272	1.000000	-0.063996	0.186306	2

In [520...]

`df.tail()`

Out[520...]

	homeshop	storeshop	employ	income	location	size	age	Cluster
employ	0.024863	0.066341	1.000000	-0.235544	0.040185	-0.241091	0.522935	1
income	-0.028314	-0.029724	-0.235544	1.000000	0.028272	0.206271	-0.111635	0
location	-0.031369	-0.000752	0.040185	0.028272	1.000000	-0.063996	0.186306	2
size	-0.085414	-0.028723	-0.241091	0.206271	-0.063996	1.000000	-0.427604	0
age	0.024872	0.089951	0.522935	-0.111635	0.186306	-0.427604	1.000000	1

**THE END**