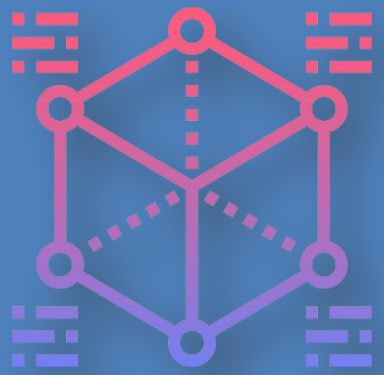
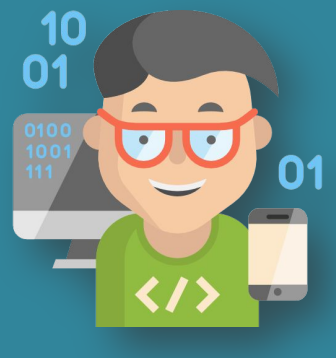




Python Programming

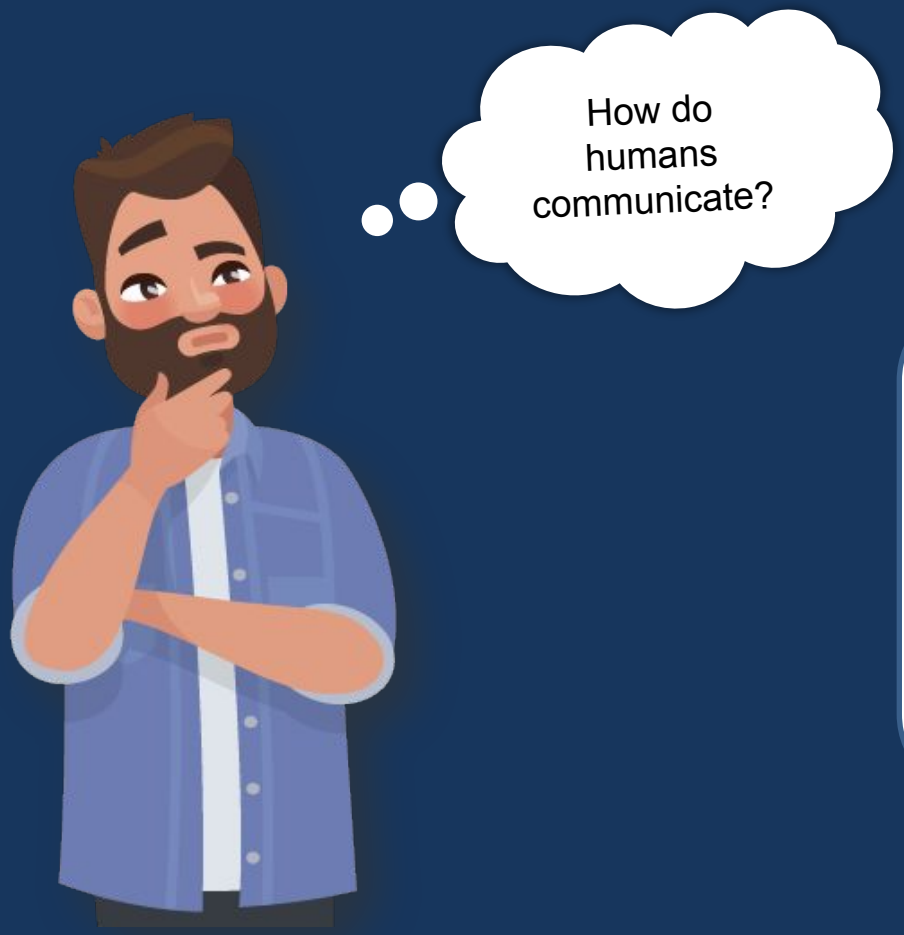
Agenda

1. SQL Interview Questions
2. Python Interview Questions
3. Data Science Interview Questions
4. NLP Interview Questions

Agenda

1. Logistic Regression
2. Logistic Regression Demo

How do Humans Communicate?



hello

Olá

Bonjour

नमस्कार

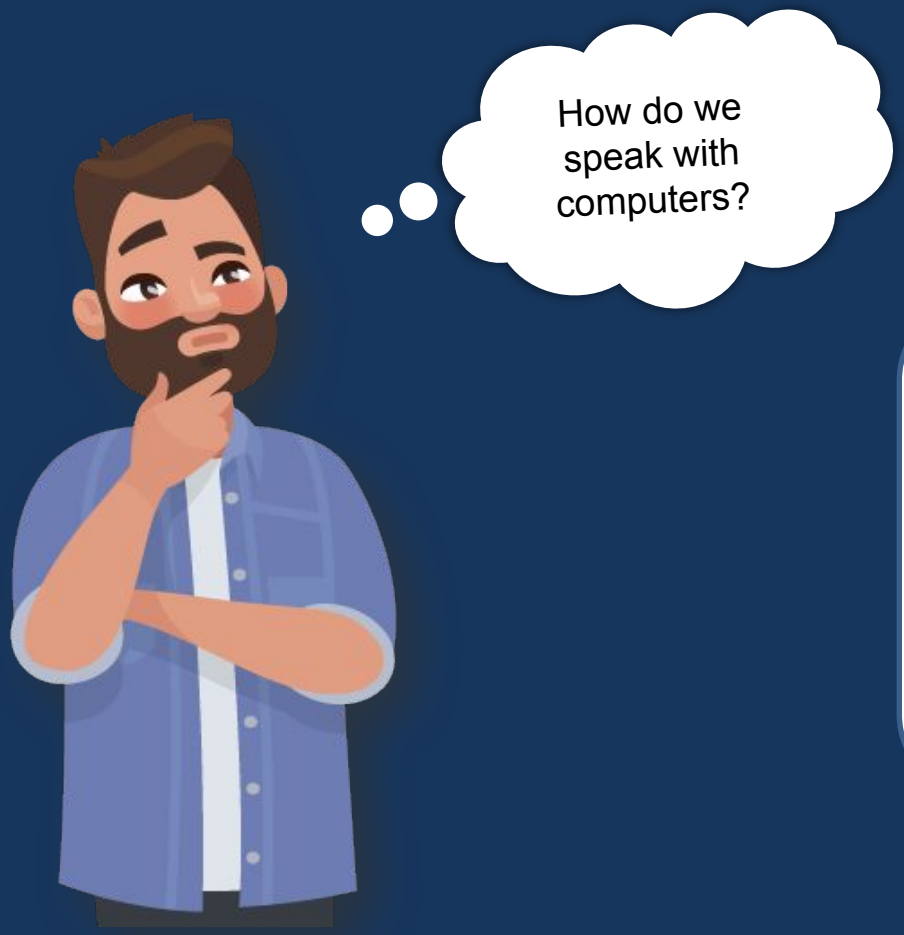
Grammar in Language

Every Language has Grammar associated with it

I am Sam

Am Sam I

Language for Computers



Java

Python

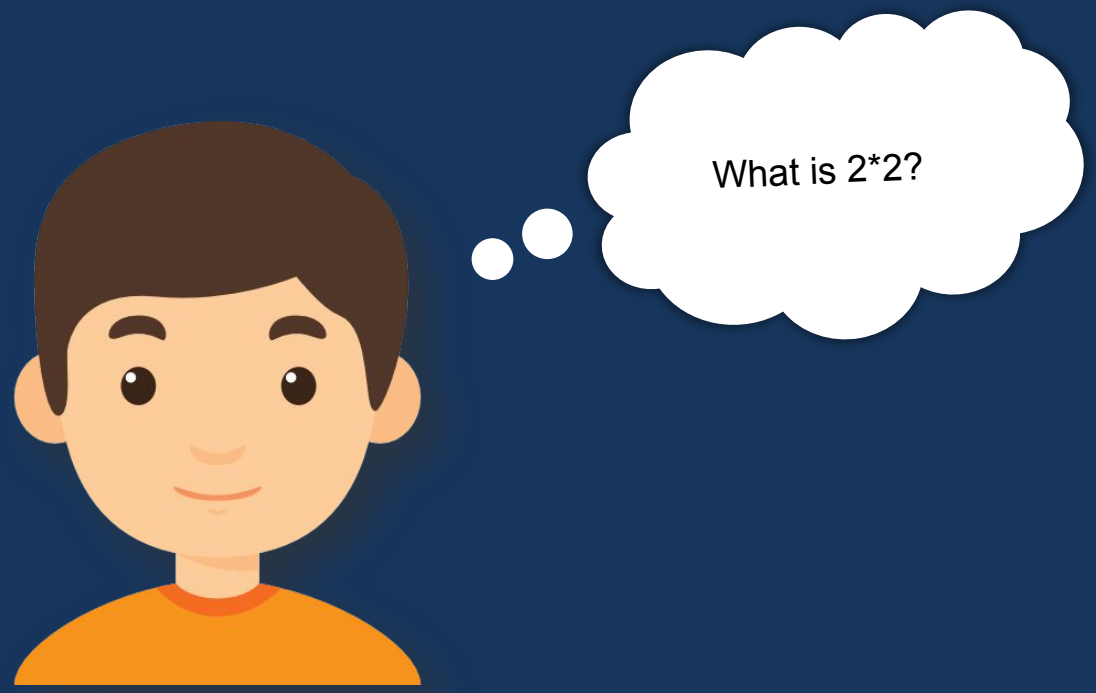
C++

Syntax for Computer Language

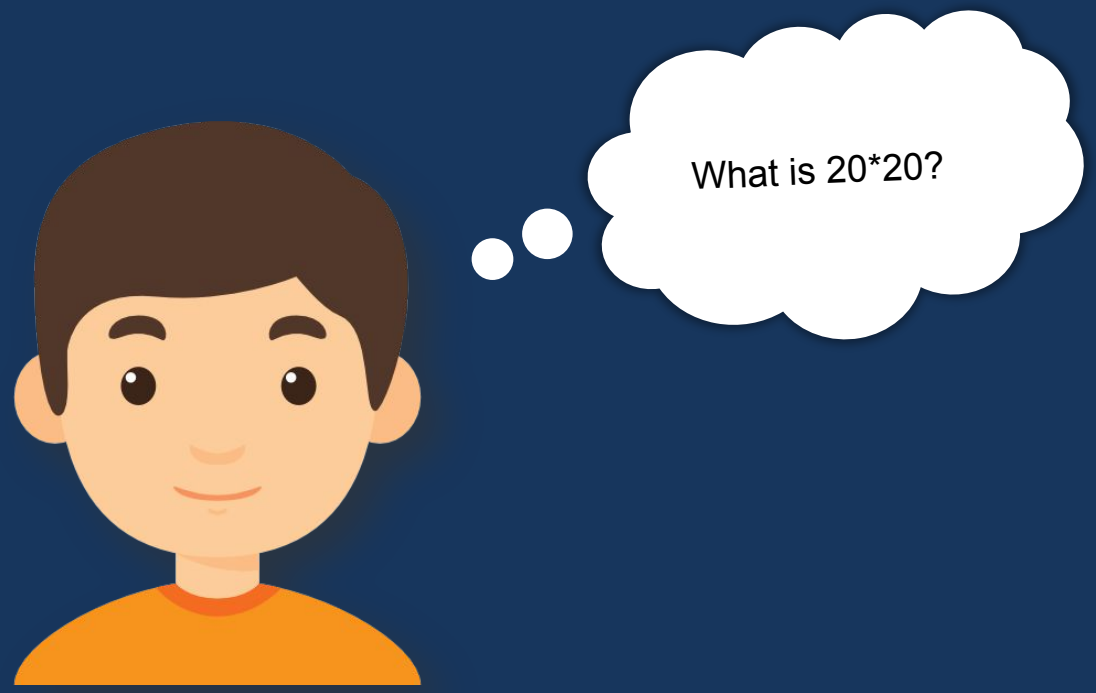
import pandas as pd

pandas import pd as

Why do we need Programming?



Why do we need Programming?



Why do we need Programming?

Applications of Programming Languages

Gaming

Banking

Machine Learning

What is Data?

13.4

287

My Name is Sam

$(a+b)^2$

0 1

How to Store Data?

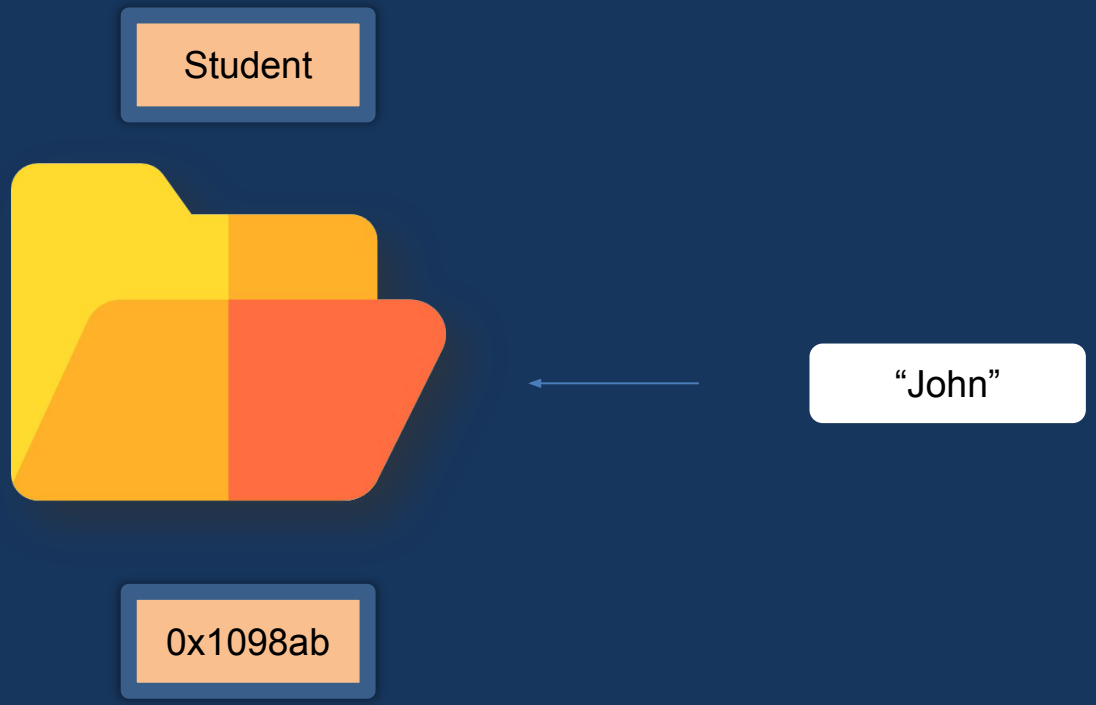
"John"

123

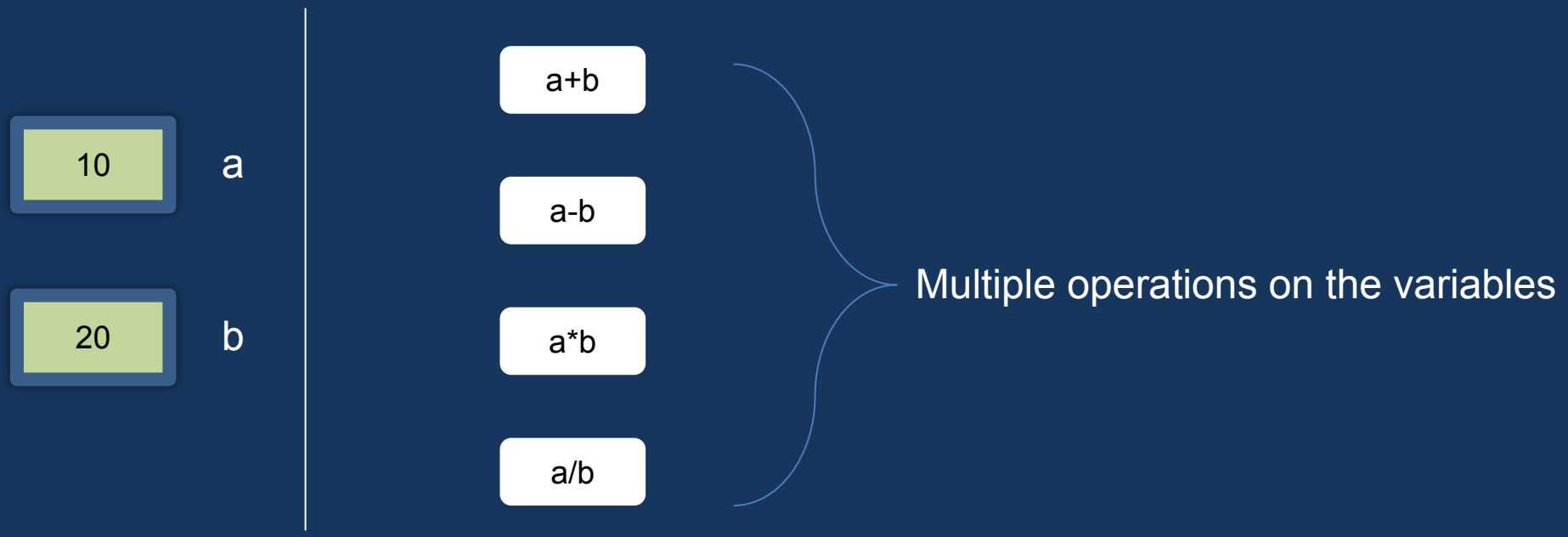
TRUE

Need of Variables

Data/Values can be stored in temporary storage spaces called variables



Example of Variable



Decision Making Statements

If
It’s raining:
Sit inside

else
Go out and Play Football

Decision Making Statements

If
Marks > 70:
Get Ice-cream

else
Give Practice Test

if…else Pseudo Code

```
If(condition){
Statements to be executed….
}

else{
Statements to be executed….
}
```

Looping Statements

Looping statements are used to repeat a task multiple times

Looping Statements

Looping Statements

While Loop Pseudo Code

```
while(TRUE){
Keep executing statements….
}
```

Functions in Real Life

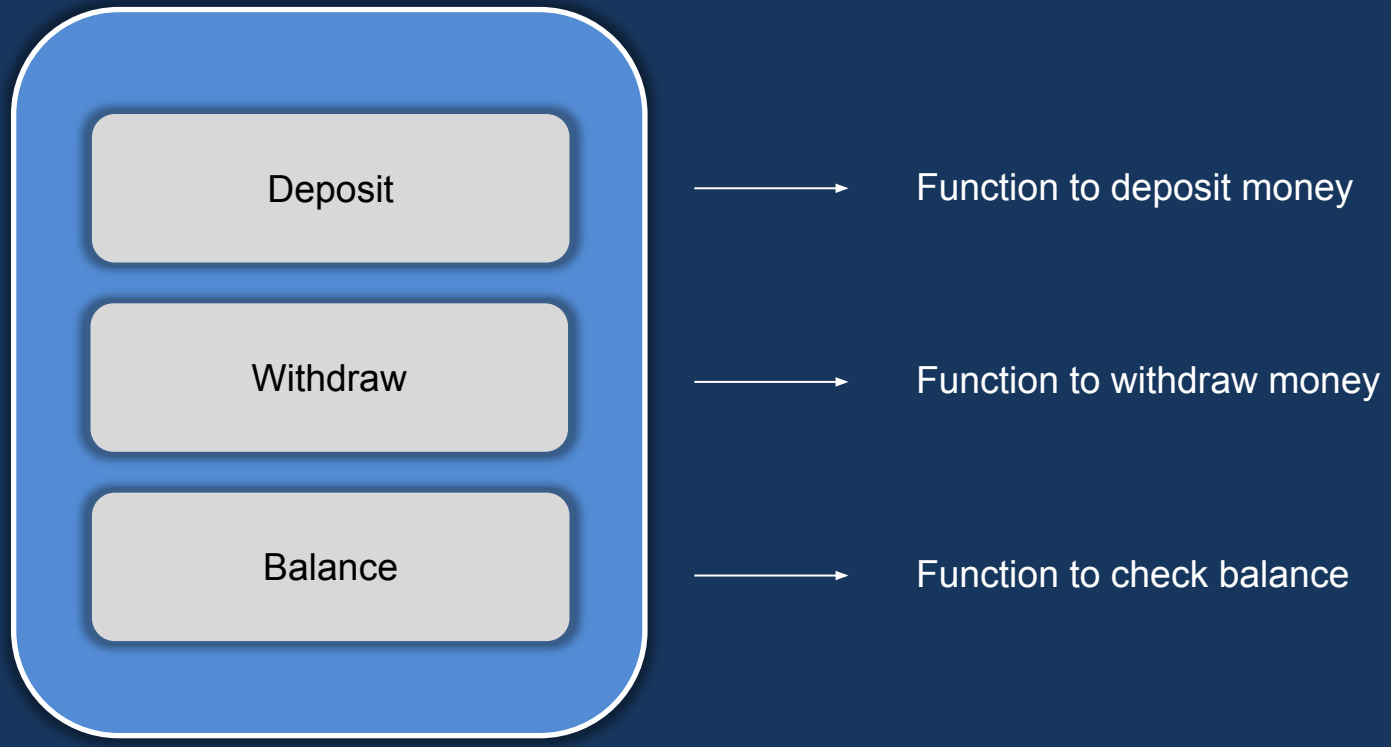
Eating

Running

Cycling

Functions in Programming World

Function is a block of code which performs a specific task



Object Oriented Programming

Classes

Class is a template/blue-print for real-world entities

Properties

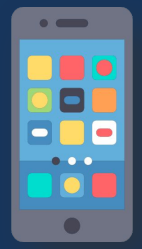
- Color
- Cost
- Battery Life

Behavior

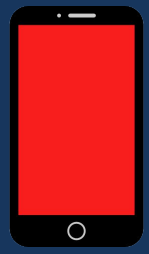
- Make Calls
- Watch Videos
- Play Games

Objects

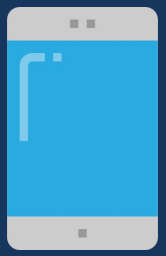
Objects are specific instances of a class



Apple



Motorola



Samsung

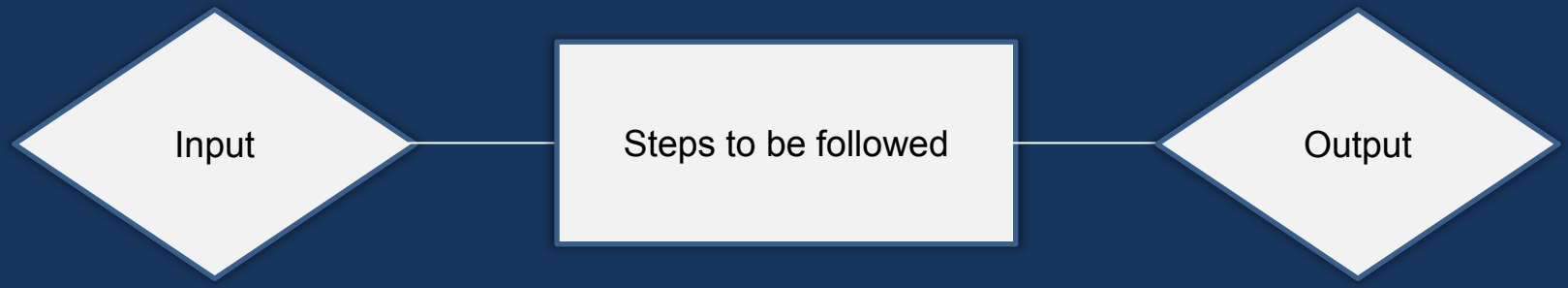
How do you solve a problem?

Step by Step Approach

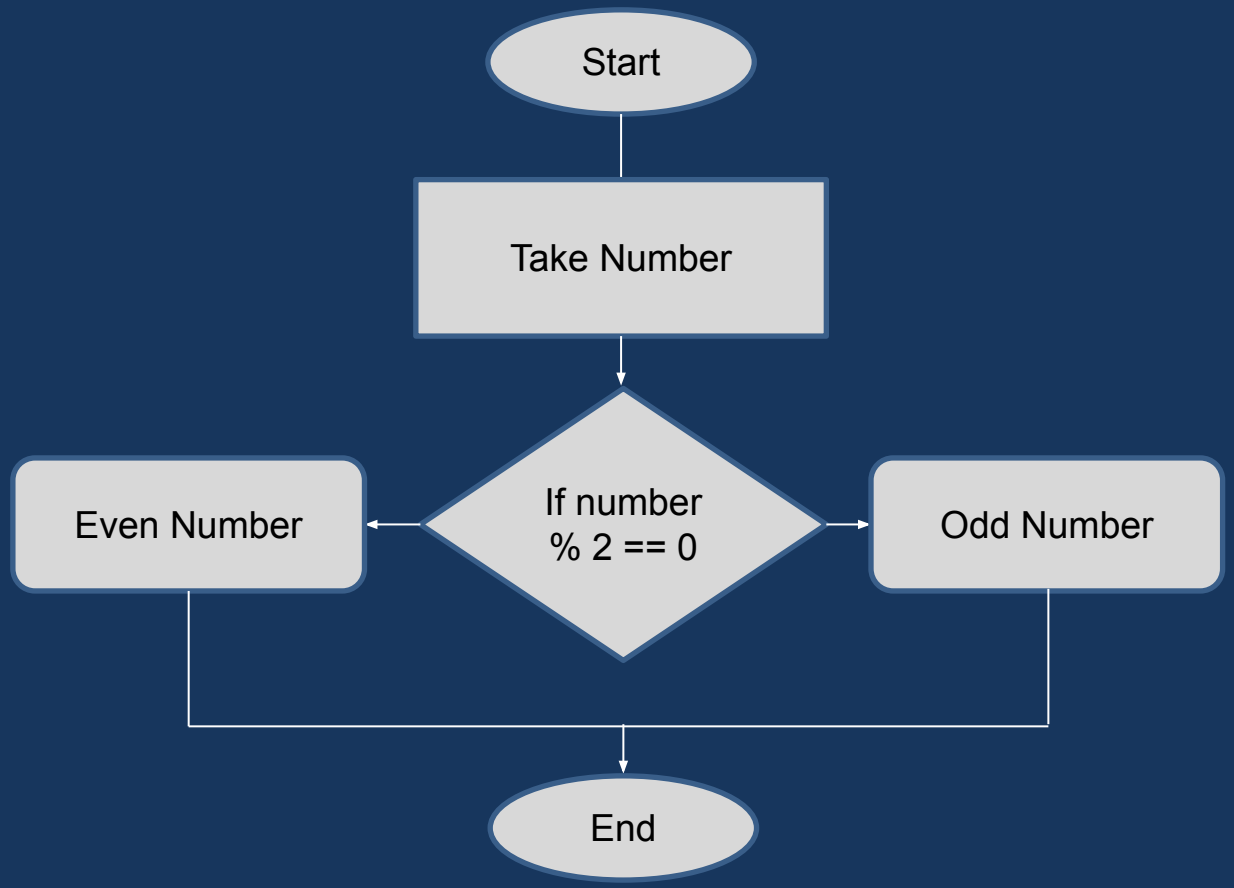
1. Check if you have all the ingredients
2. Take lemon and cut into two halves
3. Squeeze lemon into a glass of water
4. Add sugar and stir it well
5. Serve it cold with ice cubes

What is an Algorithm?

Step by step approach to solve a problem is known as algorithm



Algorithm to find if number is even/odd



Introduction to Python

Cross-Platform Compatible

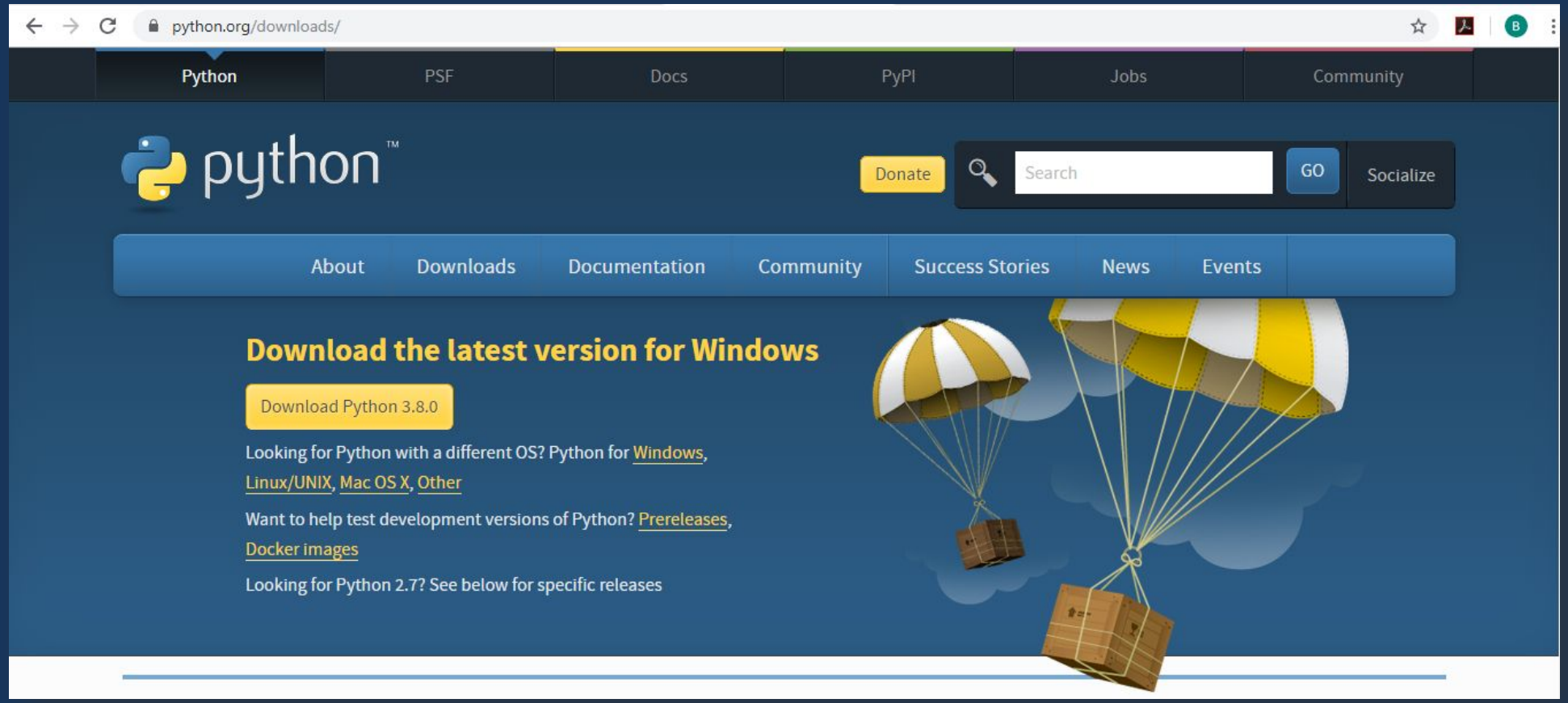
Free & Open Source

Large Standard Library

Object Oriented

Installing Python

This is the site to install Python ->

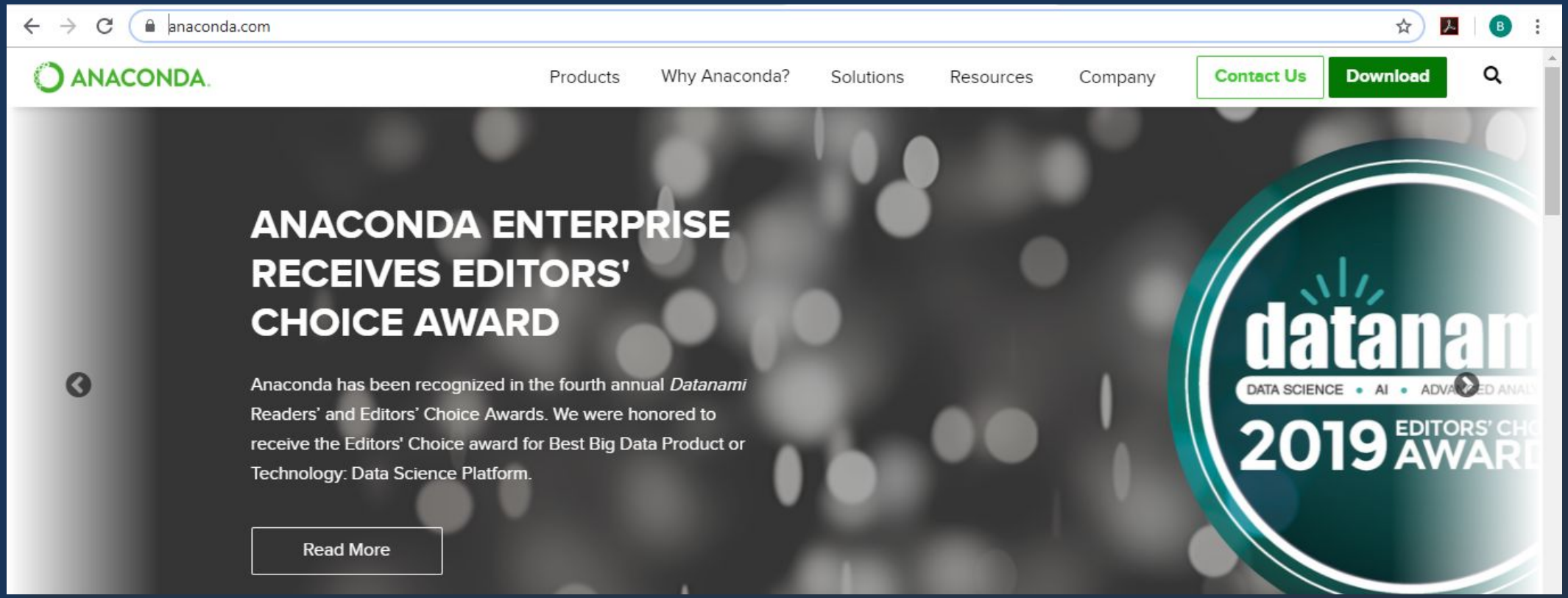


Installing PyCharm

This is the site to install PyCharm ->

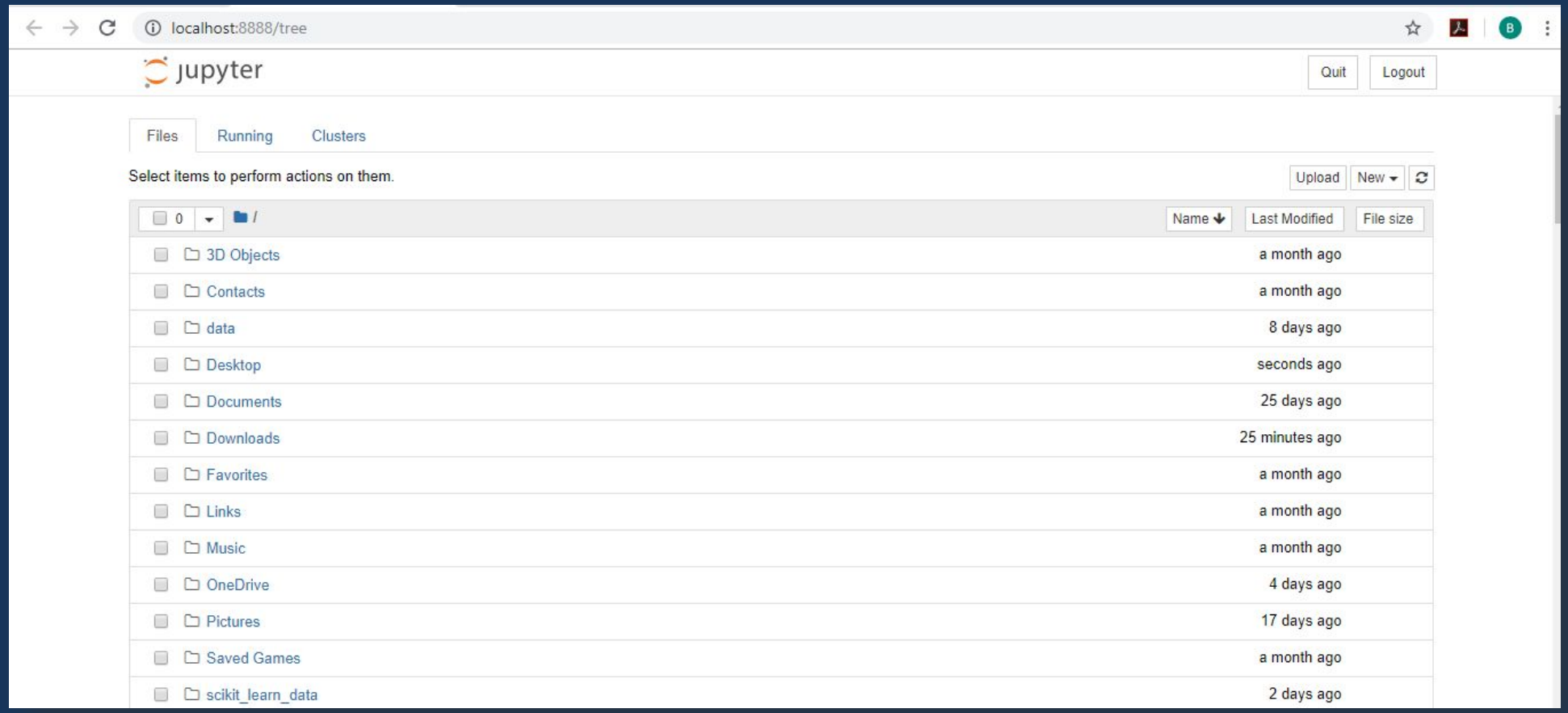
Installing Anaconda

This is the site to install Anaconda ->



Intro to Jupyter Notebook

Jupyter Notebook is a browser-based interpreter that allows us to interactively work with Python



Variables in Python

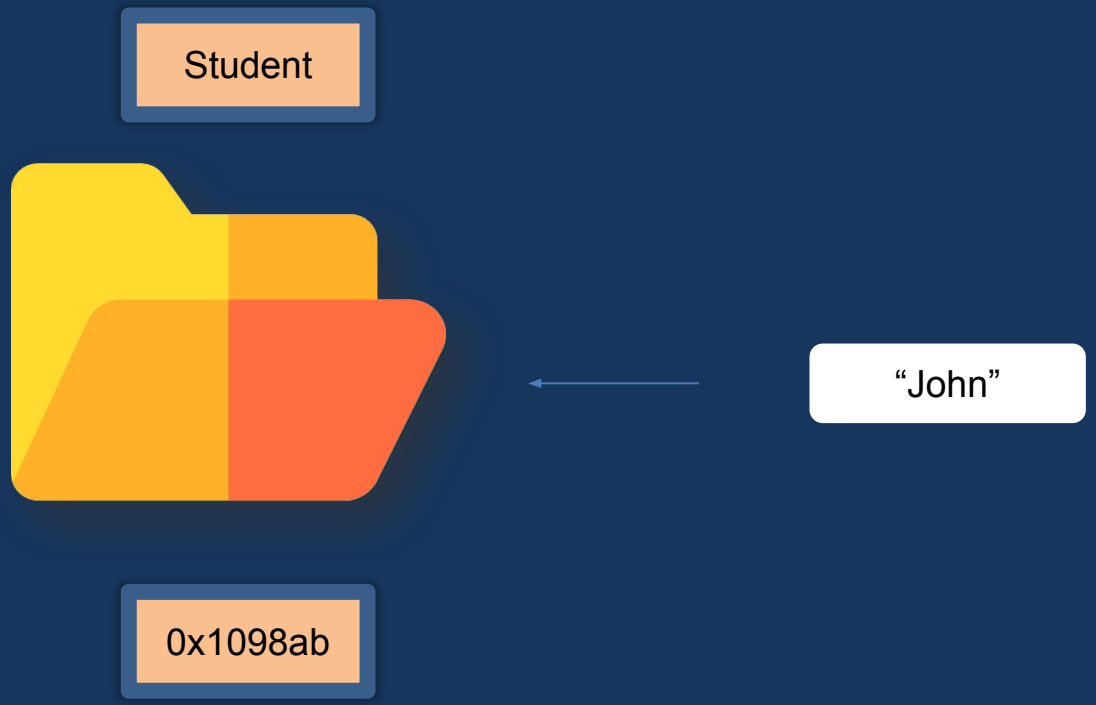
"John"

"Sam"

"Matt"

Variables in Python

Data/Values can be stored in temporary storage spaces called variables



Variables in Python

Data/Values can be stored in temporary storage spaces called variables

DataTypes in Python

Every variable is associated with a data-type

10, 500	3.14, 15.97	TRUE, FALSE	“Sam”, “Matt”
int	float	Boolean	String

Operators in Python

Arithmetic Operators

Relational Operators

Logical Operators

Python Tokens

Smallest meaningful Component in a Program

- Keywords
- Identifiers
- Literals
- Operators

Python Keywords

Keywords are special reserved words

False	class	Finally	Is	Return
None	continue	For	Lambda	Try
True	def	From	Nonlocal	While
and	del	Global	Not	With
as	elif	If	Or	Yield

Python Identifiers

Identifiers are names used for variables, functions or objects

Rules

- No special character expect _(underscore)
- Identifiers are case sensitive
- First Letter cannot be a digit

Python Literals

Literals are constants in Python

Python Strings

Strings are sequence of characters enclosed within single quotes(‘ ’), double quotes(“ “) or triple quotes(‘’’ ‘’’)

‘Hello World’

“This is Sparta”

‘’’ I am going to
France tomorrow’’’

Extracting Individual Characters

```
In [23]: my_string="My name is John"

In [24]: my_string[0]
Out[24]: 'M'
```

```
In [5]: my_string="My name is John"

In [6]: my_string[-1]
Out[6]: 'n'
```

String Functions

Finding length of string

```
In [28]: len(my_string)
Out[28]: 15
```

Converting String to lower case

```
In [30]: my_string.lower()
Out[30]: 'my name is john'
```

Converting String to upper case

```
In [31]: my_string.upper()
Out[31]: 'MY NAME IS JOHN'
```

String Functions

Replacing a substring

```
In [33]: my_string.replace('y','a')
Out[33]: 'Ma name is John'
```

Number of occurrences of substring

```
In [7]: new_string = "hello hello world"

In [8]: new_string.count("hello")
Out[8]: 2
```

String Functions

Finding the index of substring

```
In [13]: s1 = 'This is sparta!!!'
         s1.find('sparta')

Out[13]: 8
```

Splitting a String

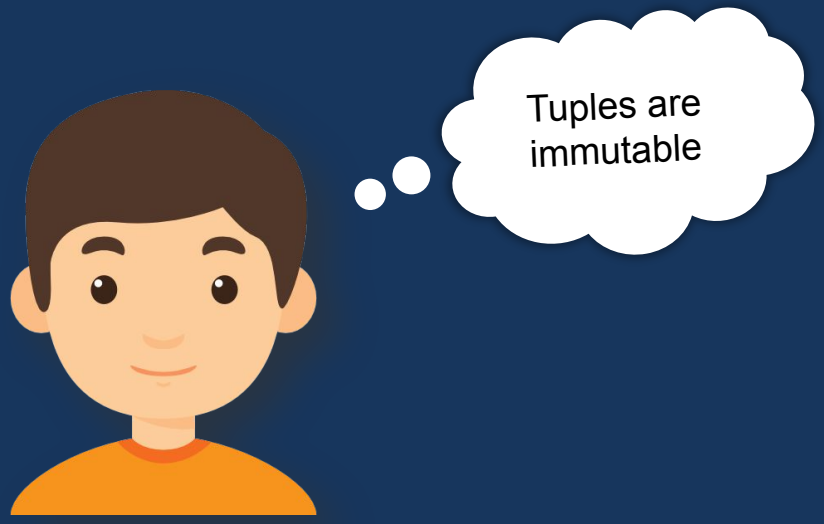
```
In [15]: fruit = 'I like apples, mangoes, bananas'

         fruit.split(',')

Out[15]: ['I like apples', ' mangoes', ' bananas']
```

Data-Structures in Python

- Tuple
- List
- Dictionary
- Set

Tuple in Python

Tuple is an ordered collection of elements enclosed within ()




```
tup1=(1,'a',True)
```

Extracting Individual Elements

```
In [20]: tup1=(1,"a",True,2,"b",False)
         tup1[0]

Out[20]: 1
```

```
In [21]: tup1=(1,"a",True,2,"b",False)
         tup1[-1]

Out[21]: False
```

```
In [22]: tup1=(1,"a",True,2,"b",False)
         tup1[1:4]

Out[22]: ('a', True, 2)
```

Modifying a Tuple

You cannot modify a tuple because it is immutable

```
In [49]: tup1[2]="hello"

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-49-2fc16622751e> in <module>
----> 1 tup1[2]="hello"

TypeError: 'tuple' object does not support item assignment
```

Tuple Basic Operations

Finding Length of Tuple

```
In [24]: tup1=(1,"a",True,2,"b",False)
         len(tup1)

Out[24]: 6
```

Concatenating Tuples

```
In [25]: tup1 = (1,2,3)
         tup2 = (4,5,6)
         tup1+tup2

Out[25]: (1, 2, 3, 4, 5, 6)
```

Tuple Basic Operations

Repeating Tuple Elements

```
In [29]: tup1 = ('sparta',300)
         tup1*3

Out[29]: ('sparta', 300, 'sparta', 300, 'sparta', 300)
```

Repeating and Concatenating

```
In [31]: tup1 = ('sparta',300)
         tup2 = (4,5,6)
         tup1*3 + tup2

Out[31]: ('sparta', 300, 'sparta', 300, 'sparta', 300, 4, 5, 6)
```

Tuple Functions

Minimum Value

```
In [32]: tup1=(1,2,3,4,5)
         min(tup1)

Out[32]: 1
```

Maximum Value

```
In [33]: tup1=(1,2,3,4,5)
         max(tup1)

Out[33]: 5
```

List in Python

List is an ordered collection of elements enclosed within []


```
l1=[1,'a',True]
```

Extracting Individual Elements

```
In [58]: l1=[1,"a",2,"b",3,"c"]
         l1[1]

Out[58]: 'a'
```

```
In [59]: l1=[1,"a",2,"b",3,"c"]
         l1[2:5]

Out[59]: [2, 'b', 3]
```

Modifying a List

Changing the element at 0th index

```
In [35]: l1=[1,"a",2,"b",3,"c"]
         l1[0]=100
         l1

Out[35]: [100, 'a', 2, 'b', 3, 'c']
```

Popping the last element

```
In [37]: l1=[1,"a",2,"b",3,"c"]
         l1.pop()
         l1

Out[37]: [1, 'a', 2, 'b', 3]
```

Appending a new element

```
In [36]: l1=[1,"a",2,"b",3,"c"]
         l1.append("Sparta")
         l1

Out[36]: [1, 'a', 2, 'b', 3, 'c', 'Sparta']
```

Modifying a List

Reversing elements of a list

```
In [40]: l1=[1,"a",2,"b",3,"c"]
         l1.reverse()
         l1

Out[40]: ['c', 3, 'b', 2, 'a', 1]
```

Inserting element at a specified index

```
In [41]: l1=[1,"a",2,"b",3,"c"]
         l1.insert(1,"Sparta")
         l1

Out[41]: [1, 'Sparta', 'a', 2, 'b', 3, 'c']
```

Sorting a list

```
In [43]: l1 = ["mango","banana","guava","apple"]
         l1.sort()
         l1

Out[43]: ['apple', 'banana', 'guava', 'mango']
```

List Basic Operations

Concatenating Lists

```
In [44]: l1 = [1,2,3]
         l2 = ["a","b","c"]
         l1+l2

Out[44]: [1, 2, 3, 'a', 'b', 'c']
```

Repeating elements

```
In [45]: l1 = [1,"a",True]
         l1*3

Out[45]: [1, 'a', True, 1, 'a', True, 1, 'a', True]
```

Dictionary in Python

Dictionary is an unordered collection of key-value pairs enclosed with {}


```
Fruit={"Apple":10,"Orange":20}
```

Extracting Keys and Values

Extracting Keys

```
In [1]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40}
        fruit.keys()

Out[1]: dict_keys(['Apple', 'Orange', 'Banana', 'Guava'])
```

Extracting Values

```
In [70]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40}
         fruit.values()

Out[70]: dict_values([10, 20, 30, 40])
```

Modifying a Dictionary

Adding a new element

```
In [2]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40}
        fruit["Mango"]=50
        fruit

Out[2]: {'Apple': 10, 'Orange': 20, 'Banana': 30, 'Guava': 40, 'Mango': 50}
```

Changing an existing element

```
In [3]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40,"Mango":50}
        fruit["Apple"]=100
        fruit

Out[3]: {'Apple': 100, 'Orange': 20, 'Banana': 30, 'Guava': 40, 'Mango': 50}
```

Dictionary Functions

Update one dictionary's elements with another

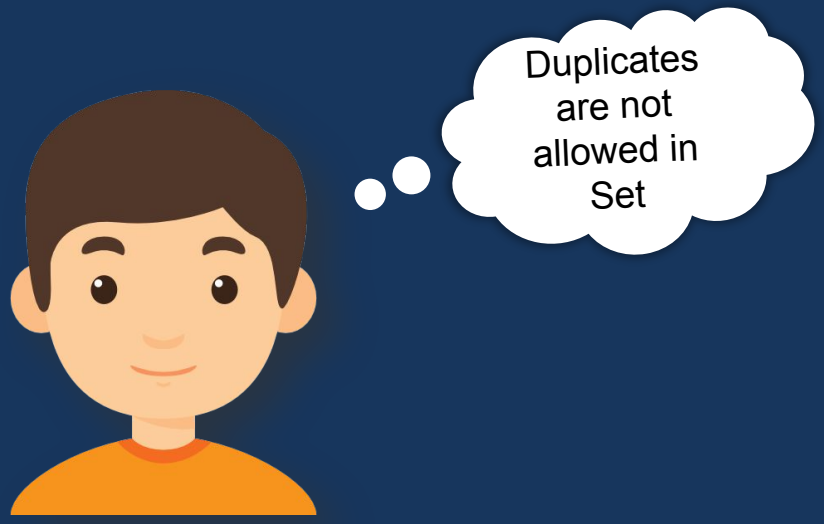
```
In [4]: fruit1={"Apple":10,"Orange":20}
        fruit2={"Banana":30,"Guava":40}

        fruit1.update(fruit2)

        fruit1

Out[4]: {'Apple': 10, 'Orange': 20, 'Banana': 30, 'Guava': 40}
```

Popping an element

```
In [6]: fruit={"Apple":10,"Orange":20,"Banana":30,"Guava":40}
        fruit.pop("Orange")
        fruit

Out[6]: {'Apple': 10, 'Banana': 30, 'Guava': 40}
```

Set in Python

Set is an unordered and unindexed collection of elements enclosed with {}




```
s1={1,"a",True}
```

Set Operations

Update one dictionary's elements with another

```
In [7]: s1={1,"a",True,2,"b",False}
        s1.add("Hello")
        s1

Out[7]: {1, 2, False, 'Hello', 'a', 'b'}
```

Removing an element

```
In [9]: s1={1,"a",True,2,"b",False}
        s1.remove("b")
        s1

Out[9]: {1, 2, False, 'a'}
```

Updating multiple elements

```
In [8]: s1={1,"a",True,2,"b",False}
        s1.update([10,20,30])
        s1

Out[8]: {1, 10, 2, 20, 30, False, 'a', 'b'}
```

Set Functions

Union of two sets

```
In [11]: s1 = {1,2,3}
         s2 = {"a","b","c"}

         s1.union(s2)

Out[11]: {1, 2, 3, 'a', 'b', 'c'}
```

Intersection of two sets

```
In [13]: s1 = {1,2,3,4,5,6}
         s2 = {5,6,7,8,9}

         s1.intersection(s2)

Out[13]: {5, 6}
```

If Statement

If
It’s raining:
Sit inside

else
Go out and Play Football

If Statement

If
Marks > 70:
Get Ice-cream

else
Give Practice Test

if…else Pseudo Code

```
If(condition){
Statements to be executed….
}

else{
Statements to be executed….
}
```

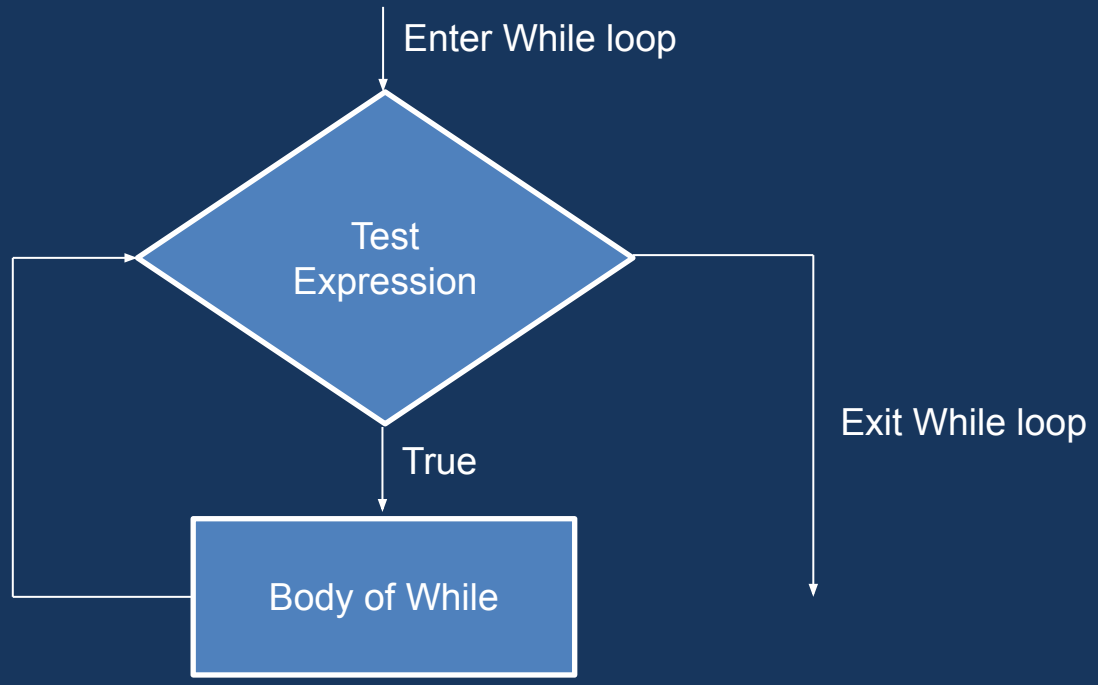
Looping Statements

Looping statements are used to repeat a task multiple times

Looping Statements

Looping Statements

While Loop

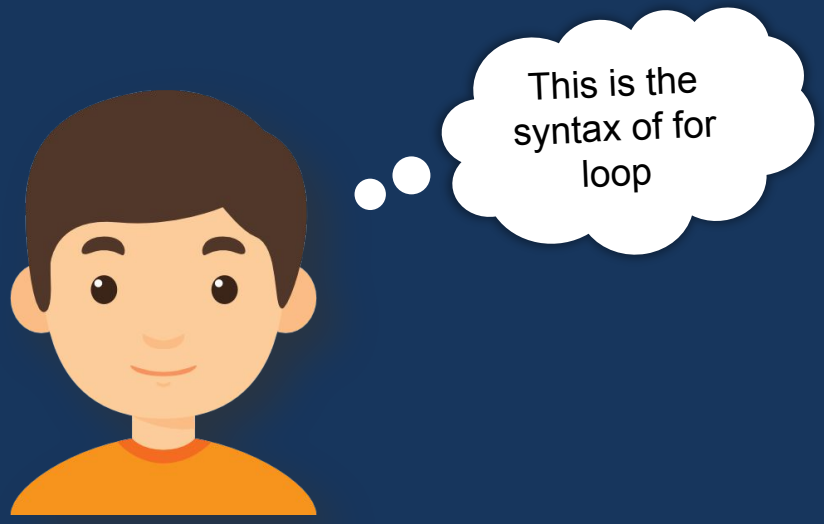


Syntax:

while condition:
    Execute Statements

For Loop

For Loop is used to iterate over a sequence(tuple, list, dictionary..)




```
for val in sequence:
    Body of for
```

Functions in Real Life

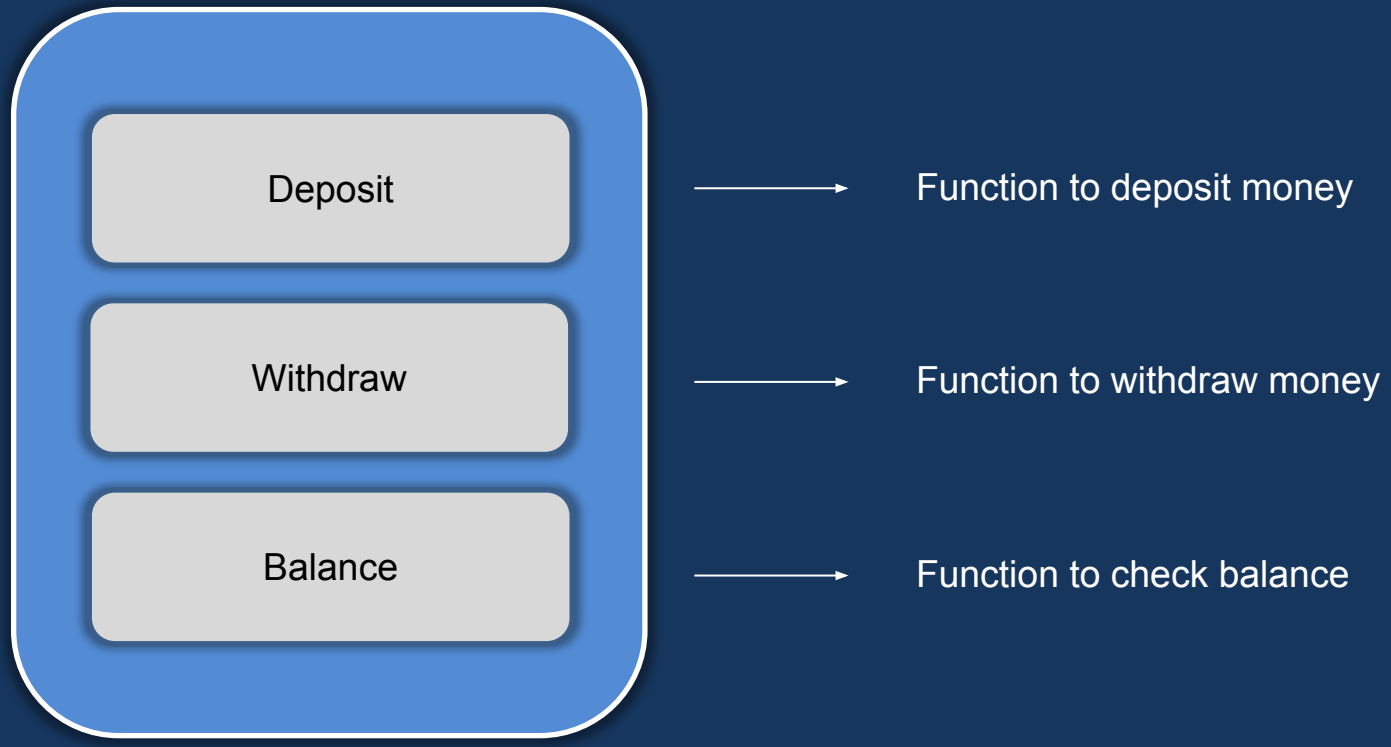
Eating

Running

Cycling

Python Functions

Function is a block of code which performs a specific task



Python Object Oriented Programming

Classes

Class is a template/blue-print for real-world entities

Properties

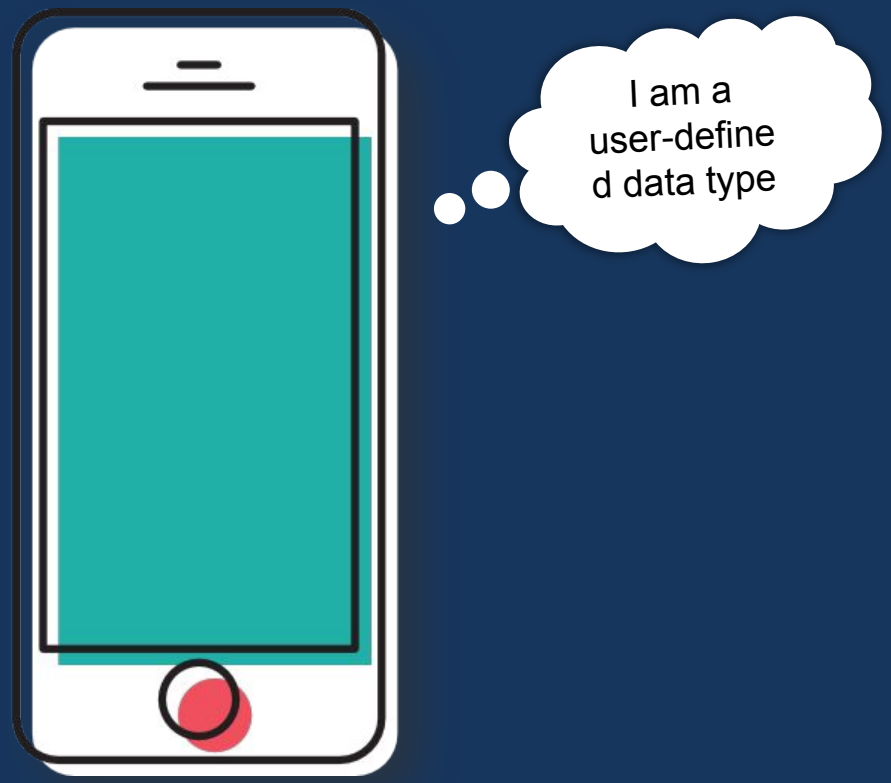
- Color
- Cost
- Battery Life

Behavior

- Make Calls
- Watch Videos
- Play Games

Class in Python

Class is a user-defined data-type



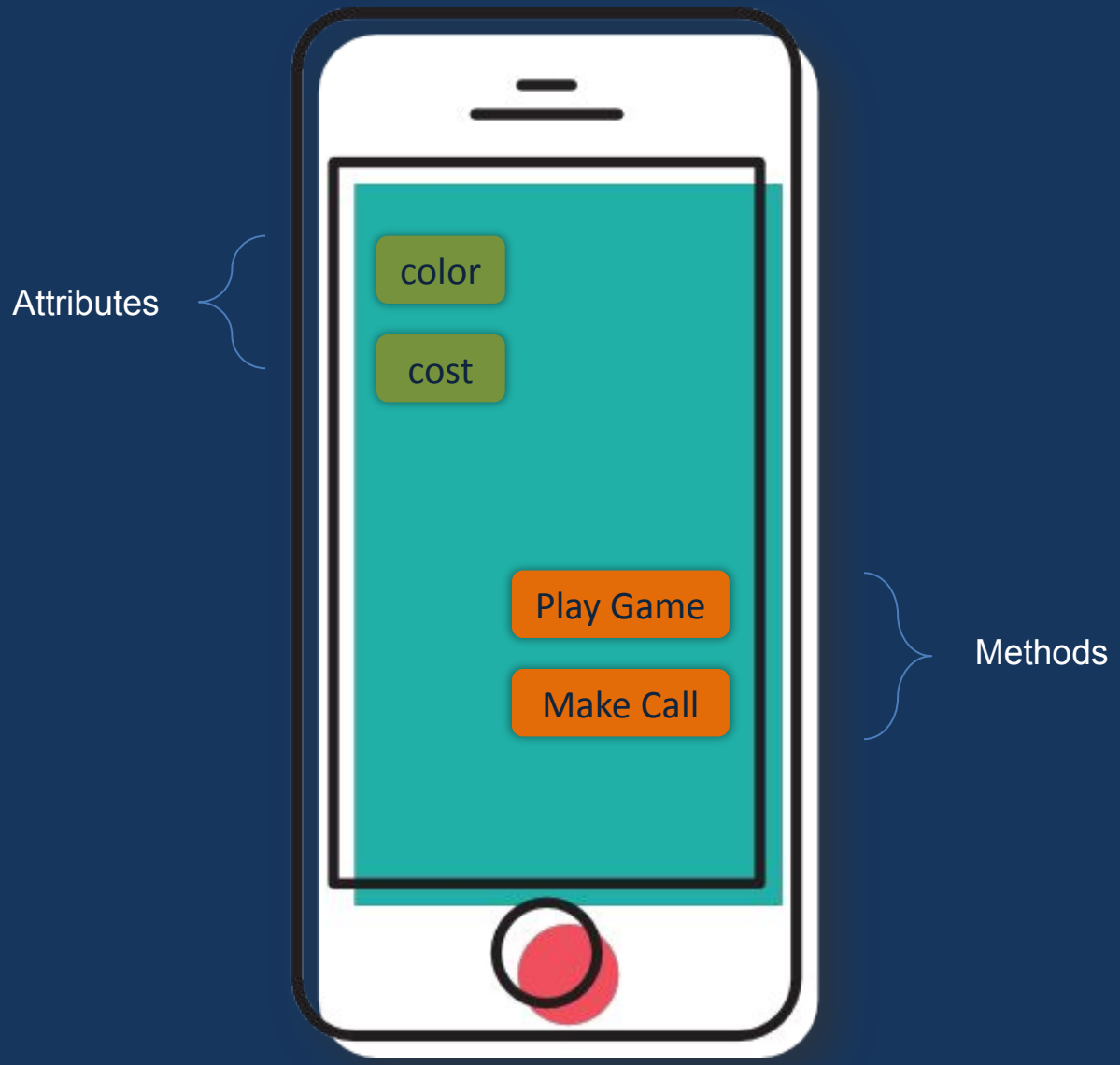
int

float

bool

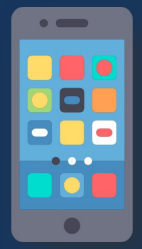
str

Attributes and Methods

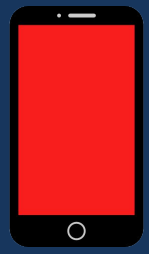


Objects

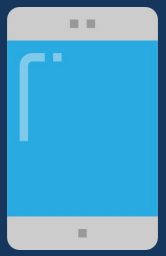
Objects are specific instances of a class



Apple



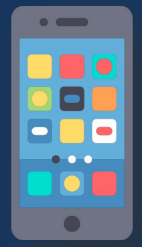
Motorola

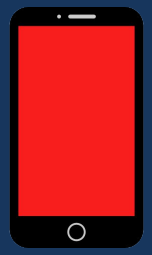


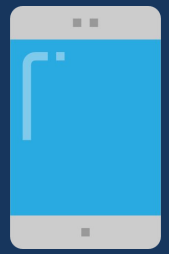
Samsung

Objects in Python

Specific instances of Mobile data type







Apple

Motorola

Samsung

---

a = 10

b = 20

c = 30

Specific instances of integer data type

Creating the first Class

```
In [1]: class Phone:

            def make_call(self):
                print("Making phone call")

            def play_game(self):
                print("Playing Game")
```

Creating the ‘Phone’ class

```
In [38]: p1=Phone()
```

Instantiating the ‘p1’ object

```
In [39]: p1.make_call()
         Making phone call

In [40]: p1.play_game()
         Playing Game
```

Invoking methods through object

Adding parameters to the class

```
In [42]: class Phone:

             def set_color(self,color):
                 self.color=color

             def set_cost(self,cost):
                 self.cost=cost

             def show_color(self):
                 return self.color

             def show_cost(self):
                 return self.cost

             def make_call(self):
                 print("Making phone call")

             def play_game(self):
                 print("Playing Game")
```

Setting and Returning the attribute values

Creating a class with Constructor

```
In [4]: class Employee:
            def __init__(self,name,age, salary,gender):

                self.name = name
                self.age = age
                self.salary =  salary
                self.gender = gender

            def employee_details(self):
                print("Name of employee is ",self.name)
                print("Age of employee is ",self.age)
                print("Salary of employee is ",self.salary)
                print("Gender of employee is ",self.gender)
```

init method acts as the constructor

Instantiating Object

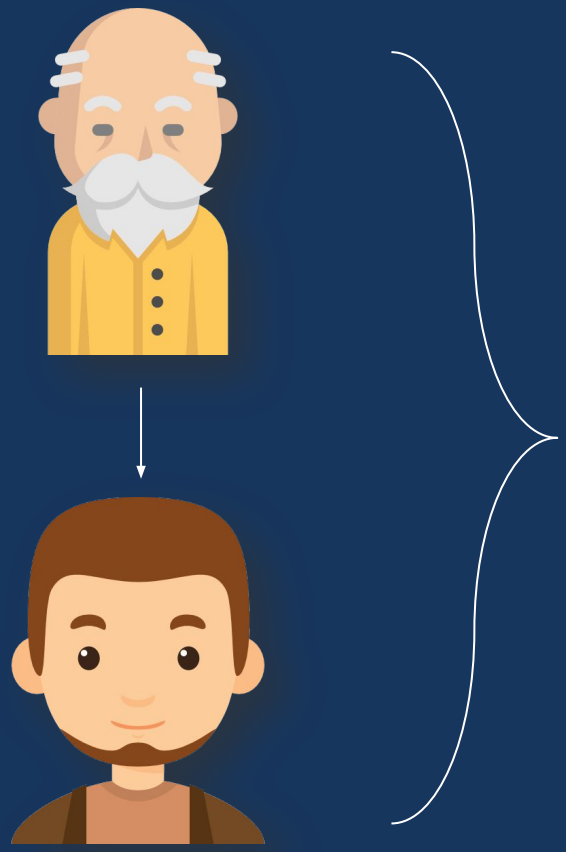
Instantiating the ‘e1’ object

Invoking the ‘employee_details’ method

```
In [5]: e1 = Employee('Sam',32,85000,'Male')

In [6]: e1.employee_details()
        Name of employee is  Sam
        Age of employee is  32
        Salary of employee is  85000
        Gender of employee is  Male
```

Inheritance in Python

With inheritance one class can derive the properties of another class



Man inheriting features from his father

Inheritance Example

```
In [23]: class Vehicle:

             def __init__(self,mileage, cost):
                 self.mileage = mileage
                 self.cost = cost

             def show_details(self):
                 print("I am a Vehicle")
                 print("Mileage of Vehicle is ", self.mileage)
                 print("Cost of Vehicle is ", self.cost)
```

Creating the base class

```
In [24]: v1 = Vehicle(500,500)
         v1.show_details()

         I am a Vehicle
         Mileage of Vehicle is  500
         Cost of Vehicle is  500
```

Instantiating the object for base class

Inheritance Example

```
In [25]: class Car(Vehicle):
             def show_car(self):
                 print("I am a car")
```

Creating the child class

```
In [26]: c1 = Car(200,1200)

In [27]: c1.show_details()

         I am a Vehicle
         Mileage of Vehicle is  200
         Cost of Vehicle is  1200
```

Instantiating the object for child class

```
In [28]: c1.show_car()

         I am a car
```

Invoking the child class method

Over-riding init method

```
In [9]: class Car(Vehicle):

            def __init__(self,mileage,cost,tyres,hp):
                super().__init__(mileage,cost)
                self.tyres = tyres
                self.hp =hp

            def show_car_details(self):
                print("I am a car")
                print("Number of tyres are ",self.tyres)
                print("Value of horse power is ",self.hp)
```

Over-riding init method

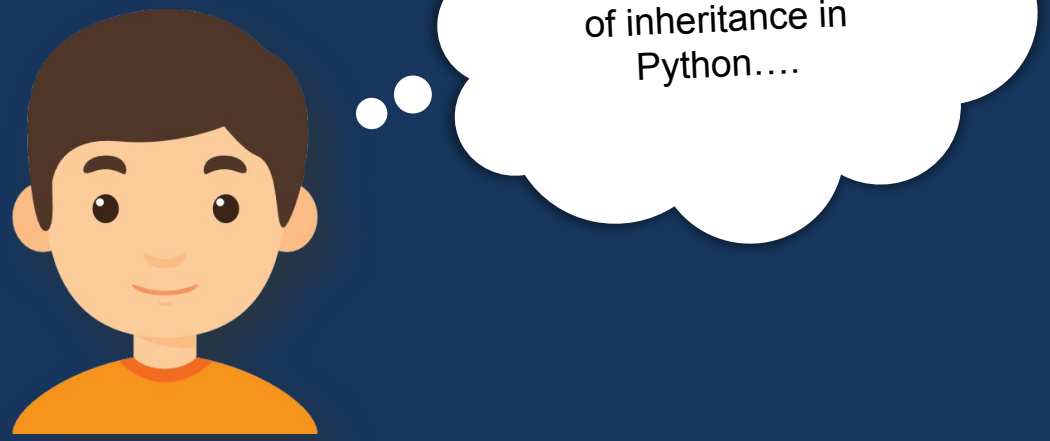
Invoking show_details() method from parent class

```
In [10]: c1 = Car(20,12000,4,300)

In [11]: c1.show_details()

         I am a Vehicle
         Mileage of Vehicle is  20
         Cost of Vehicle is  12000
```

Invoking show_car_details() method from child class

```
In [12]: c1.show_car_details()

         I am a car
         Number of tyres are  4
         Value of horse power is  300
```

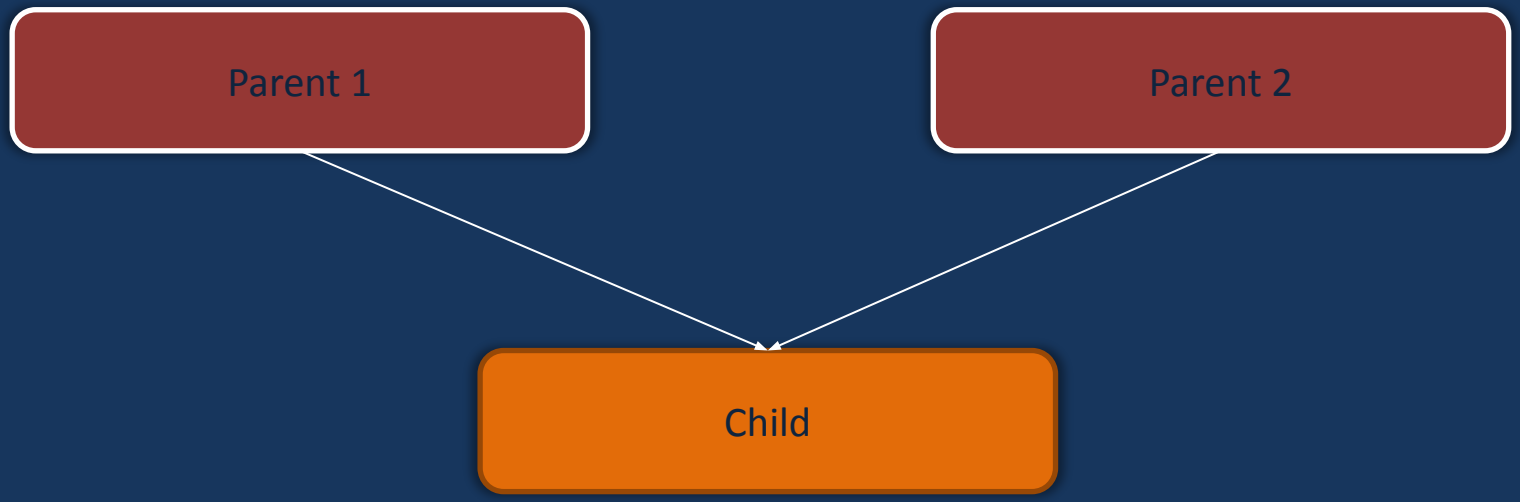
Types of Inheritance



- Single Inheritance
- Multiple Inheritance
- Multi-level Inheritance
- Hybrid Inheritance

Multiple Inheritance

In multiple inheritance, the child inherits from more than 1 parent class



Multiple Inheritance in Python

Parent Class One

```
In [35]: class Parent1():
             def assign_string_one(self,str1):
                 self.str1 = str1

             def show_string_one(self):
                 return self.str1
```

Parent Class Two

```
In [36]: class Parent2():
             def assign_string_two(self,str2):
                 self.str2 = str2

             def show_string_two(self):
                 return self.str2
```

Child Class

```
In [40]: class Derived(Parent1, Parent2):
             def assign_string_three(self,str3):
                 self.str3=str3

             def show_string_three(self):
                 return self.str3
```

Multiple Inheritance in Python

Instantiating object of child class

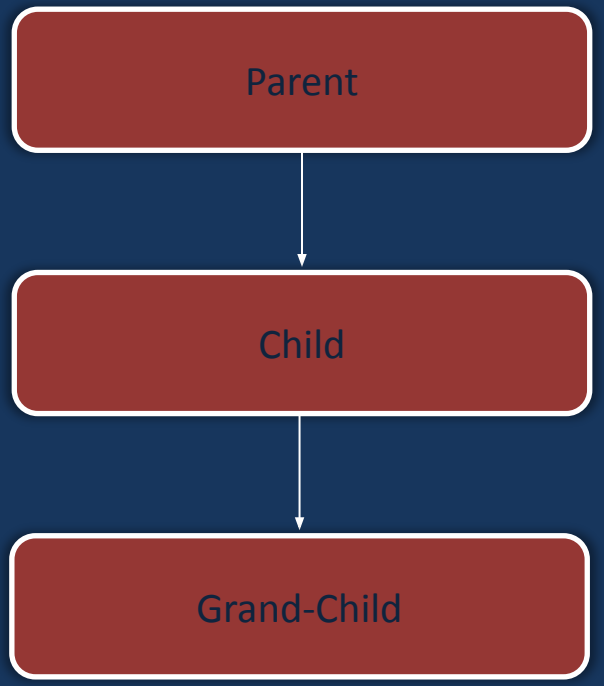
```
In [41]: d1 = Derived()

In [42]: d1.assign_string_one("one")
         d1.assign_string_two("two")
         d1.assign_string_three("three")
```

Invoking methods

```
In [46]: d1.show_string_one()
Out[46]: 'one'

In [47]: d1.show_string_two()
Out[47]: 'two'

In [48]: d1.show_string_three()
Out[48]: 'three'
```

Multi-Level Inheritance

In multi-level Inheritance, we have Parent, child, grand-child relationship



Multi-Level Inheritance in Python

Parent Class

```
In [52]: class Parent():
             def assign_name(self,name):
                 self.name = name

             def show_name(self):
                 return self.name
```

Child Class

```
In [53]: class Child(Parent):
             def assign_age(self,age):
                 self.age = age

             def show_age(self):
                 return self.age
```

Grand-Child Class

```
In [54]: class GrandChild(Child):
             def assign_gender(self,gender):
                 self.gender = gender

             def show_gender(self):
                 return self.name
```

Multi-Level Inheritance in Python

Instantiating object of GrandChild class

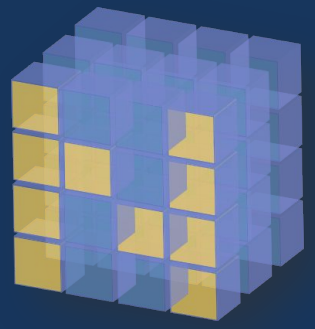
```
In [55]: g1 = GrandChild()

In [56]: g1.assign_name("Sam")
         g1.assign_age(25)
         g1.assign_gender("Male")
```

Invoking class methods

```
In [57]: g1.show_name()
Out[57]: 'Sam'

In [58]: g1.show_age()
Out[58]: 25

In [59]: g1.show_gender()
Out[59]: 'Sam'
```

Libraries in Python

Python library is a collection of functions and methods that allows you to perform many actions without writing your code



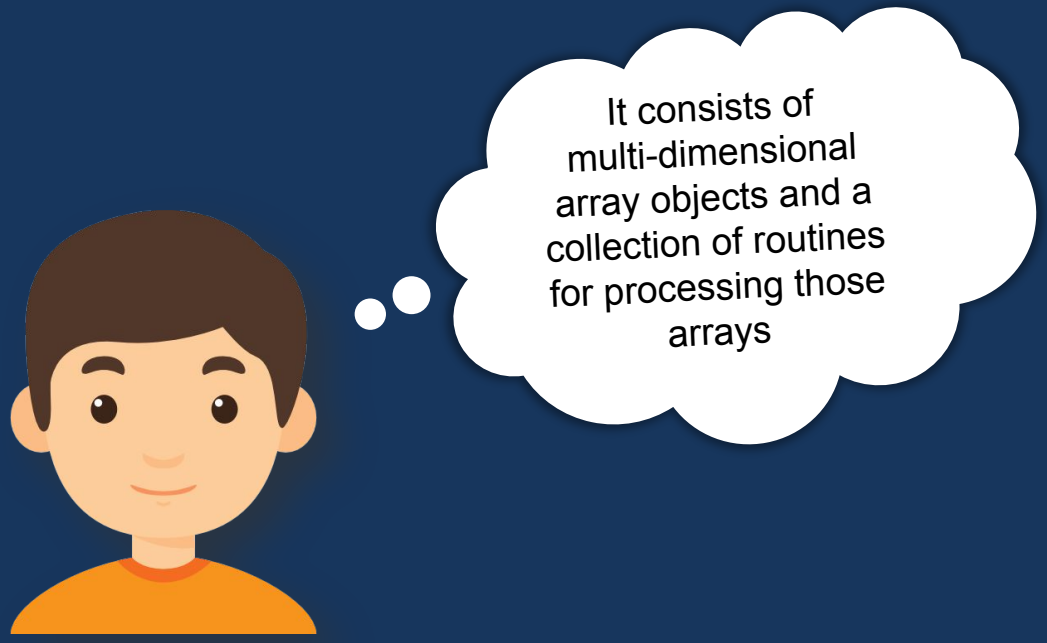
NumPy

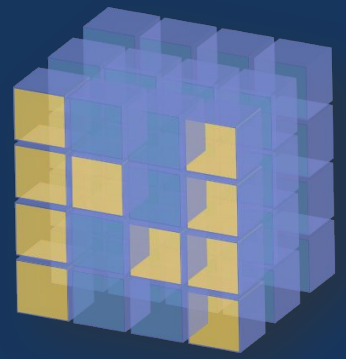
matplotlib

Pandas

Python NumPy

NumPy stands for Numerical python and is the core library for numeric and scientific computing





NumPy

Creating NumPy Array

Single-dimensional Array

```
In [3]: import numpy as np

        n1=np.array([10,20,30,40])
        n1

Out[3]: array([10, 20, 30, 40])
```

Multi-dimensional Array

```
In [6]: import numpy as np

        n2=np.array([[10,20,30,40],[40,30,20,10]])
        n2

Out[6]: array([[10, 20, 30, 40],
               [40, 30, 20, 10]])
```

Initializing NumPy Array

Initializing NumPy array with zeros

```
In [30]: import numpy as np
         n1=np.zeros((1,2))
         n1

Out[30]: array([[0., 0.]])
```

```
In [31]: import numpy as np
         n1=np.zeros((5,5))
         n1

Out[31]: array([[0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.],
                [0., 0., 0., 0., 0.]])
```

Initializing NumPy Array

Initializing NumPy array with same number

```
In [38]: import numpy as np
         n1=np.full((2,2),10)
         n1

Out[38]: array([[10, 10],
                [10, 10]])
```

Initializing NumPy Array

Initializing NumPy array within a range

```
In [34]: import numpy as np
         n1=np.arange(10,20)
         n1

Out[34]: array([10, 11, 12, 13, 14, 15, 16, 17, 18, 19])
```

```
In [35]: import numpy as np
         n1=np.arange(10,50,5)
         n1

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

Initializing NumPy Array

Initializing NumPy array with random numbers

```
In [46]: import numpy as np
         n1=np.random.randint(1,100,5)
         n1

Out[46]: array([95, 88, 26, 22, 76])
```

NumPy-Shape

Checking the shape of NumPy arrays

```
In [4]: import numpy as np
        n1=np.array([[1,2,3],[4,5,6]])
        n1.shape

Out[4]: (2, 3)
```

```
In [5]: n1.shape = (3,2)
        n1.shape

Out[5]: (3, 2)
```

Joining NumPy Arrays

vstack()

```
In [32]: import numpy as np
         n1=np.array([10,20,30])
         n2=np.array([40,50,60])

         np.vstack((n1,n2))

Out[32]: array([[10, 20, 30],
                [40, 50, 60]])
```

hstack()

```
In [33]: import numpy as np
         n1=np.array([10,20,30])
         n2=np.array([40,50,60])

         np.hstack((n1,n2))

Out[33]: array([10, 20, 30, 40, 50, 60])
```

column_stack()

```
In [34]: import numpy as np
         n1=np.array([10,20,30])
         n2=np.array([40,50,60])

         np.column_stack((n1,n2))

Out[34]: array([[10, 40],
                [20, 50],
                [30, 60]])
```

Numpy Intersection & Difference

```
In [10]: import numpy as np
         n1=np.array([10,20,30,40,50,60])
         n2=np.array([50,60,70,80,90])
```

```
In [11]: np.intersect1d(n1,n2)
Out[11]: array([50, 60])
```

```
In [10]: import numpy as np
         n1=np.array([10,20,30,40,50,60])
         n2=np.array([50,60,70,80,90])
```

```
In [23]: np.setdiff1d(n1,n2)
Out[23]: array([10, 20, 30, 40])
```

```
In [10]: import numpy as np
         n1=np.array([10,20,30,40,50,60])
         n2=np.array([50,60,70,80,90])
```

```
In [20]: np.setdiff1d(n2,n1)
Out[20]: array([70, 80, 90])
```

NumPy Array Mathematics

Addition of NumPy Arrays

```
In [13]: import numpy as np
         n1=np.array([10,20])
         n2=np.array([30,40])

         np.sum([n1,n2])

Out[13]: 100
```

```
In [14]: np.sum([n1,n2],axis=0)

Out[14]: array([40, 60])
```

```
In [15]: np.sum([n1,n2],axis=1)

Out[15]: array([30, 70])
```

NumPy Array Mathematics

Basic Addition

```
In [4]: import numpy as np
        n1=np.array([10,20,30])
        n1=n1+1
        n1

Out[4]: array([11, 21, 31])
```

Basic Multiplication

```
In [6]: import numpy as np
        n1=np.array([10,20,30])
        n1=n1*2
        n1

Out[6]: array([20, 40, 60])
```

Basic Subtraction

```
In [5]: import numpy as np
        n1=np.array([10,20,30])
        n1=n1-1
        n1

Out[5]: array([ 9, 19, 29])
```

Basic Division

```
In [7]: import numpy as np
        n1=np.array([10,20,30])
        n1=n1/2
        n1

Out[7]: array([ 5., 10., 15.])
```

NumPy Math Functions

Mean

```
In [14]: import numpy as np
         n1=np.array([10,20,30,40,50,60])
         np.mean(n1)

Out[14]: 35.0
```

Standard Deviation

```
In [17]: import numpy as np
         n1=np.array([1,5,3,100,4,48])
         np.std(n1)

Out[17]: 36.59424666377065
```

Median

```
In [16]: import numpy as np
         n1=np.array([11,44,5,96,67,85])
         np.median(n1)

Out[16]: 55.5
```

NumPy Broadcasting

Numpy Array Addition

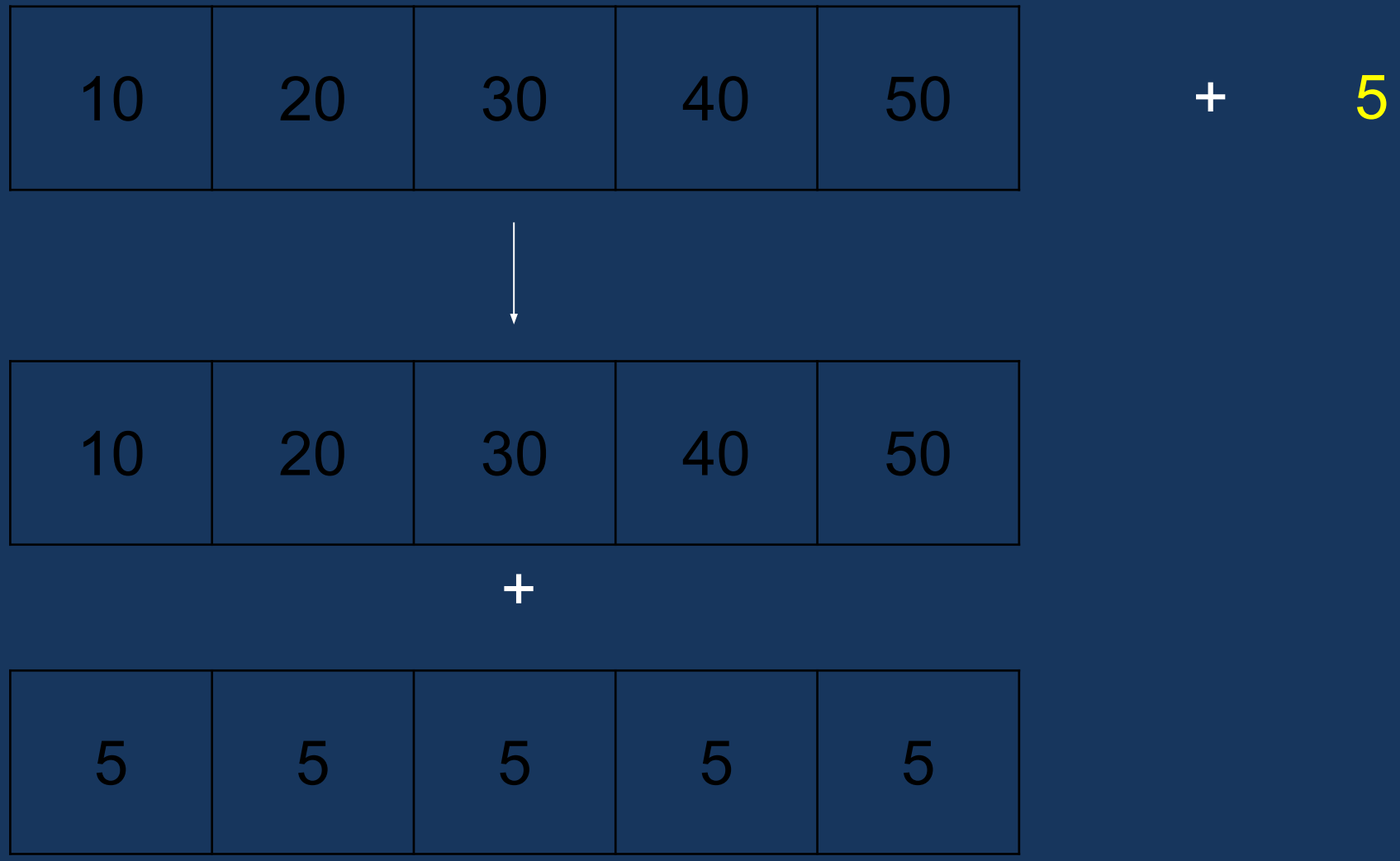
```
In [6]: import numpy as np
        n1=np.array([10,20,30,40,50,60])
        n2=np.array([50,60,70,80,90,100])
```

```
In [7]: n1 + n2
Out[7]: array([ 60,  80, 100, 120, 140, 160])
```

```
In [6]: import numpy as np
        n1=np.array([10,20,30,40,50,60])
        n2=np.array([50,60,70])
```

```
In [10]: n1 + n2

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-10-7fb1059652d4> in <module>
----> 1 n1 + n2

ValueError: operands could not be broadcast together with shapes (6,) (3,)
```

NumPy Broadcasting



NumPy Broadcasting

```
In [11]: n1=np.array([[100,200,300],[400,500,600],[700,800,900]])
         n2=np.array([10,20,30])
```



```
In [12]: n1+n2
Out[12]: array([[110, 220, 330],
                [410, 520, 630],
                [710, 820, 930]])
```

```
In [14]: n1.shape,n2.shape
Out[14]: ((3, 3), (3,))
```

NumPy Broadcasting

```
In [21]: n1 = np.array([1,2,3])
         n1.shape=(3,1)
         n1

Out[21]: array([[1],
                [2],
                [3]])
```

```
In [22]: n2 = np.array([4,5,6])
         n2.shape=(1,3)
         n2

Out[22]: array([[4, 5, 6]])
```

```
In [23]: n1 + n2

Out[23]: array([[5, 6, 7],
                [6, 7, 8],
                [7, 8, 9]])
```

NumPy Matrix

```
In [5]: import numpy as np
        n1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
        n1

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

```
In [12]: n1[0]

Out[12]: array([1, 2, 3])

In [14]: n1[1]

Out[14]: array([4, 5, 6])
```

```
In [21]: n1[:,1]

Out[21]: array([2, 5, 8])

In [22]: n1[:,2]

Out[22]: array([3, 6, 9])
```

NumPy Matrix Transpose

```
In [23]: n1

Out[23]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

```
In [24]: n1.transpose()

Out[24]: array([[1, 4, 7],
                [2, 5, 8],
                [3, 6, 9]])
```

NumPy Matrix Multiplication

```
In [25]: n1 = np.array([[1,2,3],[4,5,6],[7,8,9]])
         n1

Out[25]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])
```

```
In [26]: n2 = np.array([[9,8,7],[6,5,4],[3,2,1]])
         n2

Out[26]: array([[9, 8, 7],
                [6, 5, 4],
                [3, 2, 1]])
```

```
In [27]: n1.dot(n2)

Out[27]: array([[ 30,  24,  18],
                [ 84,  69,  54],
                [138, 114,  90]])
```

```
In [28]: n2.dot(n1)

Out[28]: array([[ 90, 114, 138],
                [ 54,  69,  84],
                [ 18,  24,  30]])
```

NumPy Save & Load

```
In [13]: import numpy as np
         n1=np.array([10,20,30,40,50,60])
         np.save('my_numpy',n1)
```

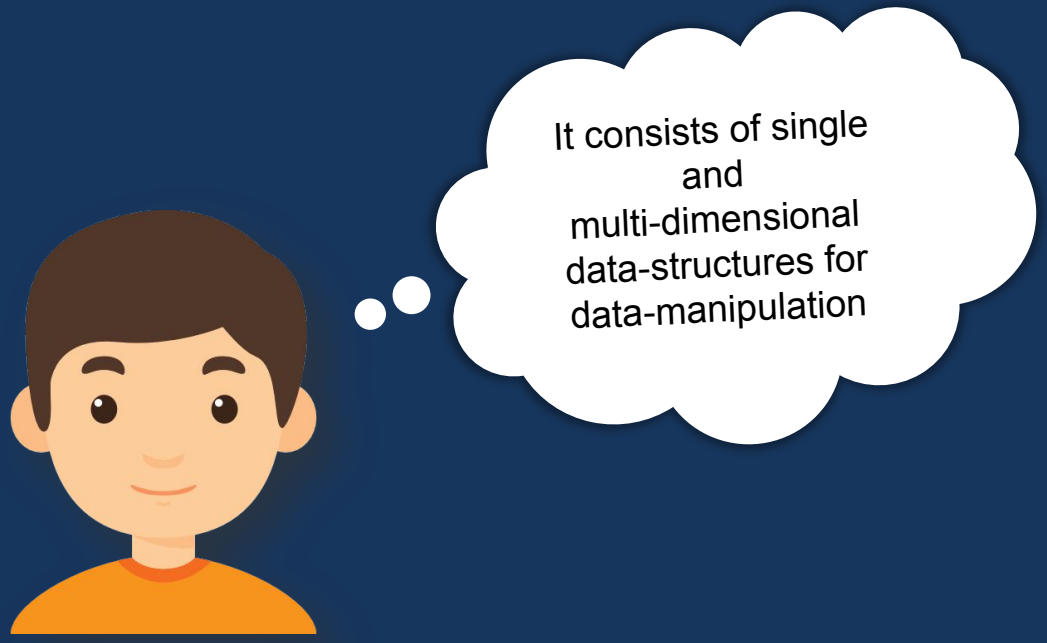
Saving Numpy Array

```
In [17]: n2=np.load('my_numpy.npy')
         n2

Out[17]: array([10, 20, 30, 40, 50, 60])
```

Loading Numpy Array

Python Pandas

Pandas stands for Panel Data and is the core library for data manipulation and data analysis



Pandas

Pandas Data-Structures

Single-dimensional

Series Object

Multi-dimensional

Data-frame

Pandas Series Object

Series Object is one-dimensional labeled array

```
In [2]: import pandas as pd
        s1=pd.Series([1,2,3,4,5])
        s1

Out[2]: 0    1
        1    2
        2    3
        3    4
        4    5
        dtype: int64
```

```
In [4]: type(s1)

Out[4]: pandas.core.series.Series
```

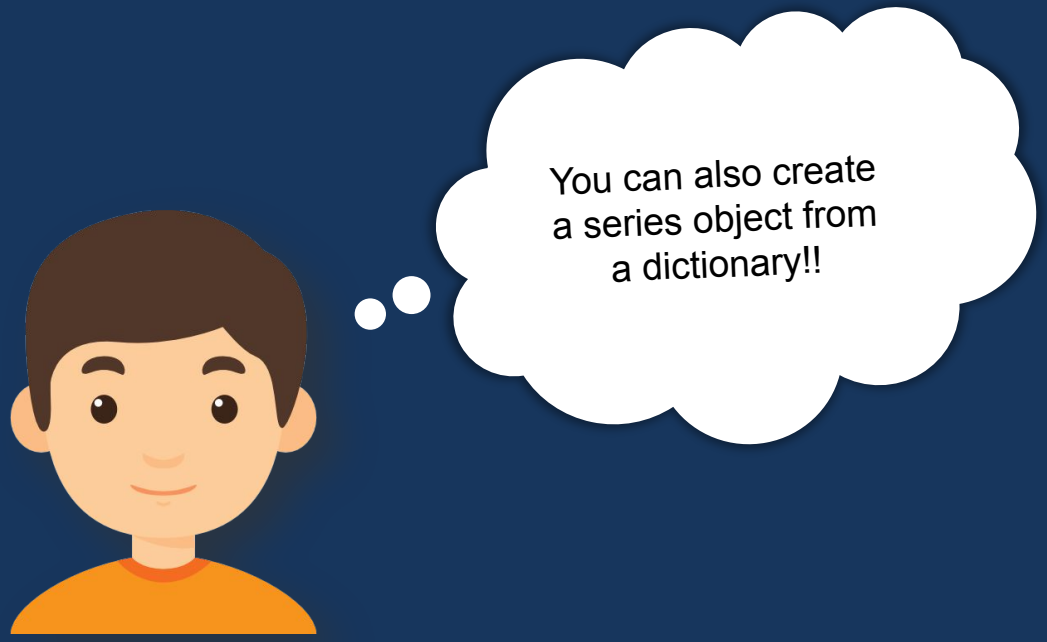
Changing Index

```
In [2]: import pandas as pd
        s1=pd.Series([1,2,3,4,5])
        s1

Out[2]: 0    1
        1    2
        2    3
        3    4
        4    5
        dtype: int64
```



```
In [5]: import pandas as pd
        s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
        s1

Out[5]: a    1
        b    2
        c    3
        d    4
        e    5
        dtype: int64
```

Series Object from Dictionary




```
In [8]: import pandas as pd
        pd.Series({'a':10,'b':20,'c':30})

Out[8]: a    10
        b    20
        c    30
        dtype: int64
```

Changing index position

```
In [6]: import pandas as pd
        pd.Series({'a':10,'b':20,'c':30},index=['b','c','d','a'])

Out[6]: b    20.0
        c    30.0
        d     NaN
        a    10.0
        dtype: float64
```

Extracting Individual Elements

Extracting a single element

```
In [15]: s1 = pd.Series([1,2,3,4,5,6,7,8,9])
         s1[3]

Out[15]: 4
```

Extracting elements from back

```
In [17]: s1 = pd.Series([1,2,3,4,5,6,7,8,9])
         s1[-3:]

Out[17]: 6    7
         7    8
         8    9
         dtype: int64
```

Extracting a sequence of elements

```
In [16]: s1 = pd.Series([1,2,3,4,5,6,7,8,9])
         s1[:4]

Out[16]: 0    1
         1    2
         2    3
         3    4
         dtype: int64
```

Basic Math Operations on Series

Adding a scalar value to Series Elements

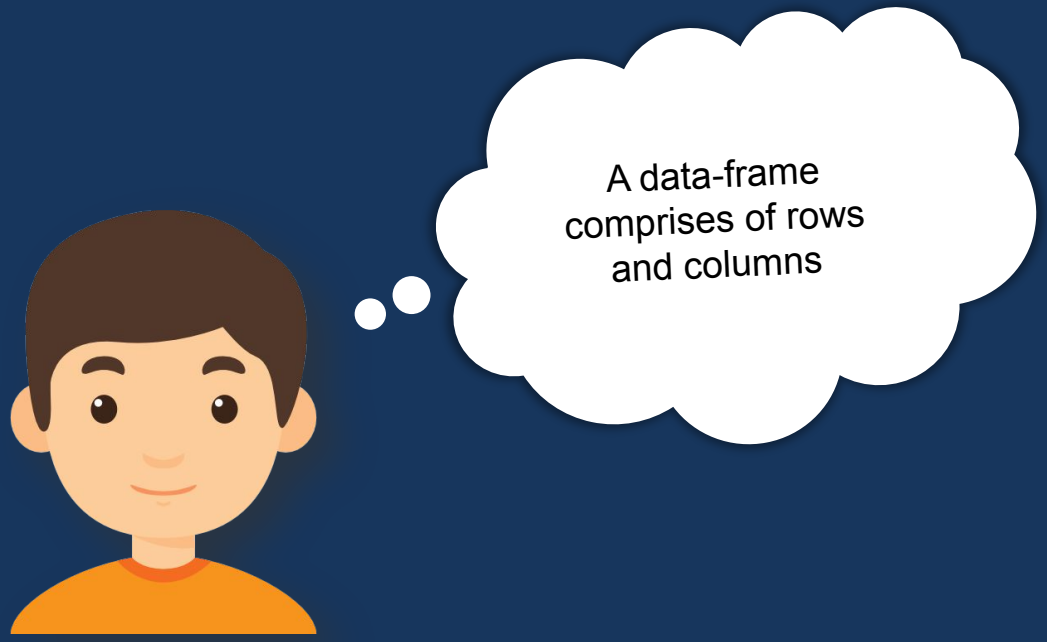
```
In [26]: s1 +5

Out[26]: 0     6
         1     7
         2     8
         3     9
         4    10
         5    11
         6    12
         7    13
         8    14
         dtype: int64
```

Adding two Series Objects

```
In [24]: s1 = pd.Series([1,2,3,4,5,6,7,8,9])
         s2 = pd.Series([10,20,30,40,50,60,70,80,90])

In [25]: s1+s2

Out[25]: 0    11
         1    22
         2    33
         3    44
         4    55
         5    66
         6    77
         7    88
         8    99
         dtype: int64
```

Pandas Dataframe

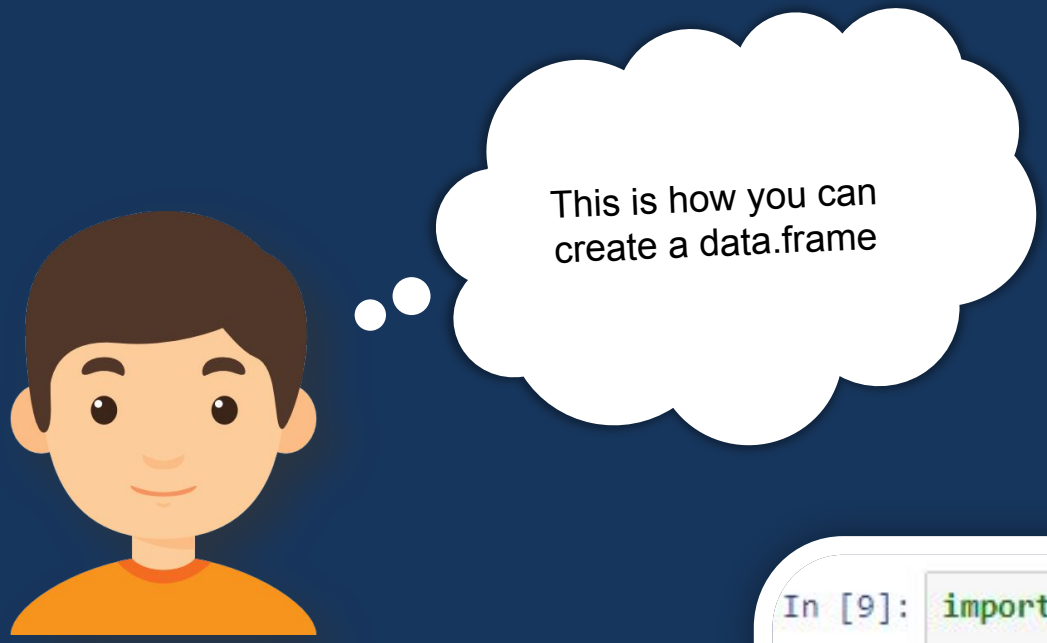
Dataframe is a 2-dimensional labelled data-structure



Out[9]:

	Name	Marks
0	Bob	76
1	Sam	25
2	Anne	92

Creating a Dataframe




```
In [9]: import pandas as pd

        pd.DataFrame({"Name":['Bob','Sam','Anne'],"Marks":[76,25,92]})
```

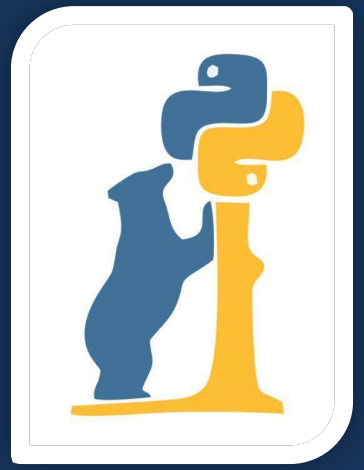
Out[9]:

	Name	Marks
0	Bob	76
1	Sam	25
2	Anne	92

Dataframe In-Built Functions

head()

shape()



describe()

tail()

.iloc[]

	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

```
iris.iloc[0:3,0:2]
```

	Sepal.Length	Sepal.Width
0	5.1	3.5
1	4.9	3.0
2	4.7	3.2

.loc[]

```
iris.loc[0:3,("Sepal.Length","Petal.Length")]
```

	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

	Sepal.Length	Petal.Length
0	5.1	1.4
1	4.9	1.4
2	4.7	1.3
3	4.6	1.5

Dropping Columns

```
iris.drop('Sepal.Length',axis=1)
```

	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

	Sepal.Width	Petal.Length	Petal.Width	Species
0	3.5	1.4	0.2	setosa
1	3.0	1.4	0.2	setosa
2	3.2	1.3	0.2	setosa
3	3.1	1.5	0.2	setosa
4	3.6	1.4	0.2	setosa

Dropping Rows

```
iris.drop([1,2,3],axis=0)
```

	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

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
5	5.4	3.9	1.7	0.4	setosa
6	4.6	3.4	1.4	0.3	setosa
7	5.0	3.4	1.5	0.2	setosa

Combining Data-Frames

Creating first dataframe

```
In [25]: df1 = pd.DataFrame({"emp_id":['101','102','103','104'],
                             'dept': ['content','content','sales','tech'],
                             'gender': ['male','female','male','male'],
                             'salary': [120000,52000,214000,163000]},index=[0,1,2,3])
```

Creating second dataframe

```
In [26]: df2 = pd.DataFrame({"emp_id":['101','103','104','105'],
                             'location': ['new york','boston','boston','New Jersey'],
                             'distance': [12,9,44,21],
                             'salary': [120000,214000,163000,331000]},index=[0,1,2,3])
```

Data-Frames Concatenation

Axis=0

```
In [27]: pd.concat([df1,df2],axis=0,sort=False)
```

Out[27]:

	emp_id	dept	gender	salary	location	distance
0	101	content	male	120000	NaN	NaN
1	102	content	female	52000	NaN	NaN
2	103	sales	male	214000	NaN	NaN
3	104	tech	male	163000	NaN	NaN
0	101	NaN	NaN	120000	new york	12.0
1	103	NaN	NaN	214000	boston	9.0
2	104	NaN	NaN	163000	boston	44.0
3	105	NaN	NaN	331000	New Jersey	21.0

Data-Frames Concatenation

Axis=0, Sort = True

```
In [28]: pd.concat([df1,df2],axis=0,sort=True)
```

Out[28]:

	dept	distance	emp_id	gender	location	salary
0	content	NaN	101	male	NaN	120000
1	content	NaN	102	female	NaN	52000
2	sales	NaN	103	male	NaN	214000
3	tech	NaN	104	male	NaN	163000
0	NaN	12.0	101	NaN	new york	120000
1	NaN	9.0	103	NaN	boston	214000
2	NaN	44.0	104	NaN	boston	163000
3	NaN	21.0	105	NaN	New Jersey	331000

Data-Frames Concatenation

Axis=1

```
In [29]: pd.concat([df1,df2],axis=1,sort=False)
```

Out[29]:

	emp_id	dept	gender	salary	emp_id	location	distance	salary
0	101	content	male	120000	101	new york	12	120000
1	102	content	female	52000	103	boston	9	214000
2	103	sales	male	214000	104	boston	44	163000
3	104	tech	male	163000	105	New Jersey	21	331000

Data-Frames Merge – Inner Join

Inner Join

```
In [33]: pd.merge(df1,df2,how='inner',on='emp_id')
```

Out[33]:

	emp_id	dept	gender	salary_x	location	distance	salary_y
0	101	content	male	120000	new york	12	120000
1	103	sales	male	214000	boston	9	214000
2	104	tech	male	163000	boston	44	163000

Data-Frames Merge – Left Join

Left Join

```
In [34]: pd.merge(df1,df2,how='left',on='emp_id')
```

Out[34]:

	emp_id	dept	gender	salary_x	location	distance	salary_y
0	101	content	male	120000	new york	12.0	120000.0
1	102	content	female	52000	NaN	NaN	NaN
2	103	sales	male	214000	boston	9.0	214000.0
3	104	tech	male	163000	boston	44.0	163000.0

Data-Frames Merge – Right Join

Right Join

```
In [35]: pd.merge(df1,df2,how='right',on='emp_id')
```

Out[35]:

	emp_id	dept	gender	salary_x	location	distance	salary_y
0	101	content	male	120000.0	new york	12	120000
1	103	sales	male	214000.0	boston	9	214000
2	104	tech	male	163000.0	boston	44	163000
3	105	NaN	NaN	NaN	New Jersey	21	331000

Data-Frames Merge – Outer Join

Outer Join

```
In [36]: pd.merge(df1,df2,how='outer',on='emp_id')
```

Out[36]:

	emp_id	dept	gender	salary_x	location	distance	salary_y
0	101	content	male	120000.0	new york	12.0	120000.0
1	102	content	female	52000.0	NaN	NaN	NaN
2	103	sales	male	214000.0	boston	9.0	214000.0
3	104	tech	male	163000.0	boston	44.0	163000.0
4	105	NaN	NaN	NaN	New Jersey	21.0	331000.0

More Pandas Functions

Mean

```
In [5]: iris.mean()

Out[5]: Sepal.Length    5.843333
        Sepal.Width     3.057333
        Petal.Length    3.758000
        Petal.Width     1.199333
        dtype: float64
```

Minimum

```
In [13]: iris.min()

Out[13]: Sepal.Length        4.3
         Sepal.Width           2
         Petal.Length          1
         Petal.Width         0.1
         Species          setosa
         dtype: object
```

Median

```
In [6]: iris.median()

Out[6]: Sepal.Length    5.80
        Sepal.Width     3.00
        Petal.Length    4.35
        Petal.Width     1.30
        dtype: float64
```

Maximum

```
In [14]: iris.max()

Out[14]: Sepal.Length         7.9
         Sepal.Width          4.4
         Petal.Length         6.9
         Petal.Width          2.5
         Species        virginica
         dtype: object
```

More Pandas Functions

	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

```
def half(s):
    return s*0.5
```

```
iris[['Sepal.Length','Petal.Length']].apply(half)
```

	Sepal.Length	Petal.Length
0	2.55	0.70
1	2.45	0.70
2	2.35	0.65
3	2.30	0.75
4	2.50	0.70

More Pandas Functions

Value_counts()

```
In [28]: iris['Species'].value_counts()

Out[28]: setosa        50
         virginica     50
         versicolor    50
         Name: Species, dtype: int64
```

sort_values()

```
In [29]: iris.sort_values(by='Sepal.Length')
```

Out[29]:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
13	4.3	3.0	1.1	0.1	setosa
42	4.4	3.2	1.3	0.2	setosa
38	4.4	3.0	1.3	0.2	setosa
8	4.4	2.9	1.4	0.2	setosa
41	4.5	2.3	1.3	0.3	setosa

Pokemon Analysis

```
In [8]: import pandas as pd
        pokemon = pd.read_csv('pokemon.csv')
        pokemon.head()
```

	abilities	against_bug	against_dark	against_dragon	against_electric	against_fairy	against_fight
0	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5
1	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5
2	['Overgrow', 'Chlorophyll']	1.0	1.0	1.0	0.5	0.5	0.5
3	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0
4	['Blaze', 'Solar Power']	0.5	1.0	1.0	1.0	0.5	1.0

Understanding Data

```
In [11]: pokemon.shape
Out[11]: (801, 41)
```

```
In [15]: pokemon.describe()
```

Out[15]:

	against_bug	against_dark	against_dragon	against_electric	against_fairy
count	801.000000	801.000000	801.000000	801.000000	801.000000
mean	0.996255	1.057116	0.968789	1.073970	1.068976
std	0.597248	0.438142	0.353058	0.654962	0.522167
min	0.250000	0.250000	0.000000	0.000000	0.250000
25%	0.500000	1.000000	1.000000	0.500000	1.000000
50%	1.000000	1.000000	1.000000	1.000000	1.000000
75%	1.000000	1.000000	1.000000	1.000000	1.000000
max	4.000000	4.000000	2.000000	4.000000	4.000000

Looking at Null Values

```
In [20]: pokemon.isnull().values.any()
Out[20]: True
```

```
In [22]: pokemon.isnull().values.sum()
Out[22]: 522
```

```
In [28]: pokemon.isnull().sum()
Out[28]: abilities          0
         against_bug        0
         against_dark       0
         against_dragon     0
         against_electric   0
         against_fairy      0
         against_fight      0
         against_fire       0
         against_flying     0
         against_ghost      0
```

Imputing Null Values

```
In [31]: pokemon['height_m'].fillna((pokemon['height_m'].mean()), inplace=True)
```

```
In [35]: pokemon['weight_kg'].fillna((pokemon['weight_kg'].median()), inplace=True)
```

```
In [39]: pokemon['percentage_male'].fillna(50, inplace=True)
```

Checking Frequency

```
In [43]: pokemon['is_legendary'].value_counts()

Out[43]: 0    731
         1     70
         Name: is_legendary, dtype: int64
```

```
In [44]: pokemon['generation'].value_counts()

Out[44]: 5    156
         1    151
         3    135
         4    107
         2    100
         7     80
         6     72
         Name: generation, dtype: int64
```

Renaming Columns

```
In [49]: pokemon.rename(columns = {'type1':'primary_type'}, inplace = True)
```

```
In [50]: pokemon.rename(columns = {'type2':'secondary_type'}, inplace = True)
```

```
In [51]: pokemon.rename(columns = {'name':'pokemon_name'}, inplace = True)
```

Extracting Primary Types

```
In [53]: grass_pokemon=pokemon[pokemon['primary_type']=="grass"]
```

```
In [56]: fire_pokemon=pokemon[pokemon['primary_type']=="fire"]
```

```
In [58]: water_pokemon=pokemon[pokemon['primary_type']=="water"]
```

Extracting Primary & Secondary Types

```
In [70]: grass_fairy_pokemon=pokemon[(pokemon['primary_type']=="grass") & (pokemon['secondary_type']=="fairy")]
```

```
In [73]: fire_fighting_pokemon=pokemon[(pokemon['primary_type']=="fire") & (pokemon['secondary_type']=="fighting")]
```

```
In [76]: water_bug_pokemon=pokemon[(pokemon['primary_type']=="water") & (pokemon['secondary_type']=="bug")]
```

Extracting Specific Pokemons

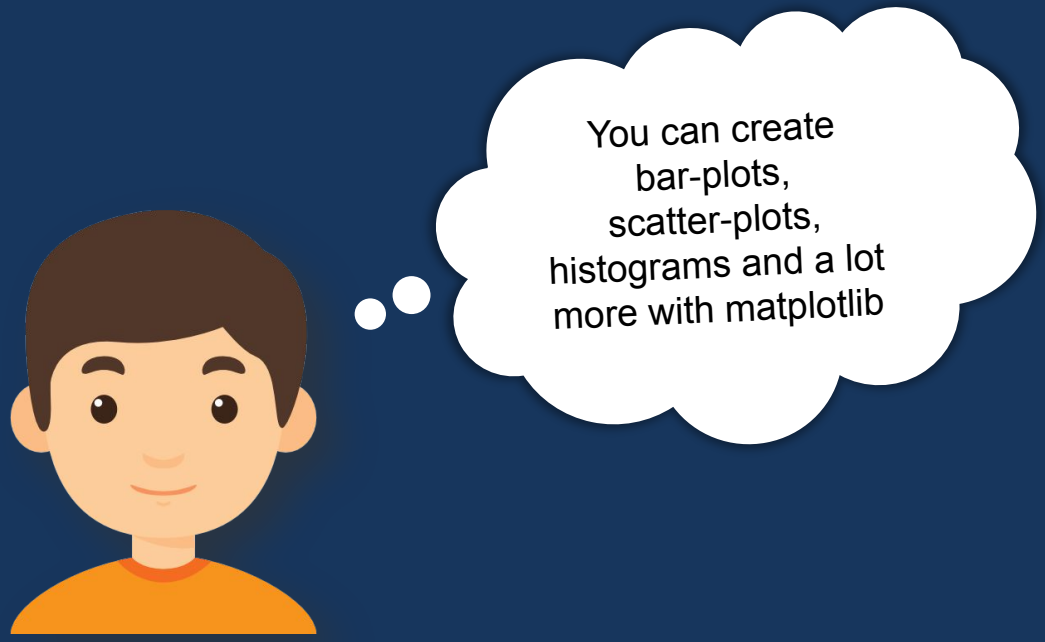
```
In [79]: high_attack_defence=pokemon[(pokemon['sp_attack']>100) & (pokemon['sp_defense']>100)]
```

```
In [83]: pokemon[(pokemon['weight_kg']>100) & (pokemon['generation']==4)]
```

```
In [84]: pokemon[(pokemon['pokedex_number']<300) & (pokemon['speed']<50)]
```

Python Matplotlib

Matplotlib is a python library used for data visualization



matplotlib

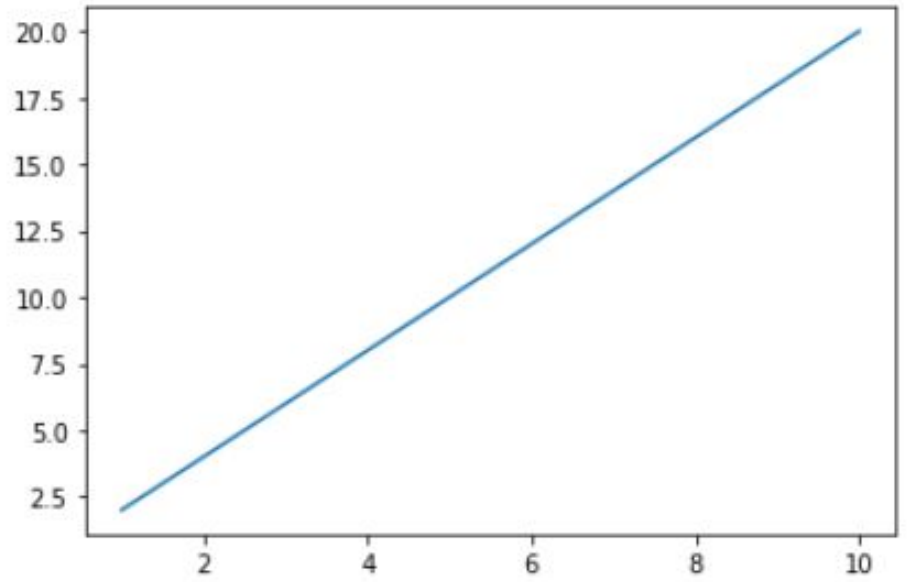
Line Plot

```
In [1]: import numpy as np
        from matplotlib import pyplot as plt
```

```
In [2]: x=np.arange(1,11)
        x
Out[2]: array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
```

```
In [4]: y= 2*x
        y
Out[4]: array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

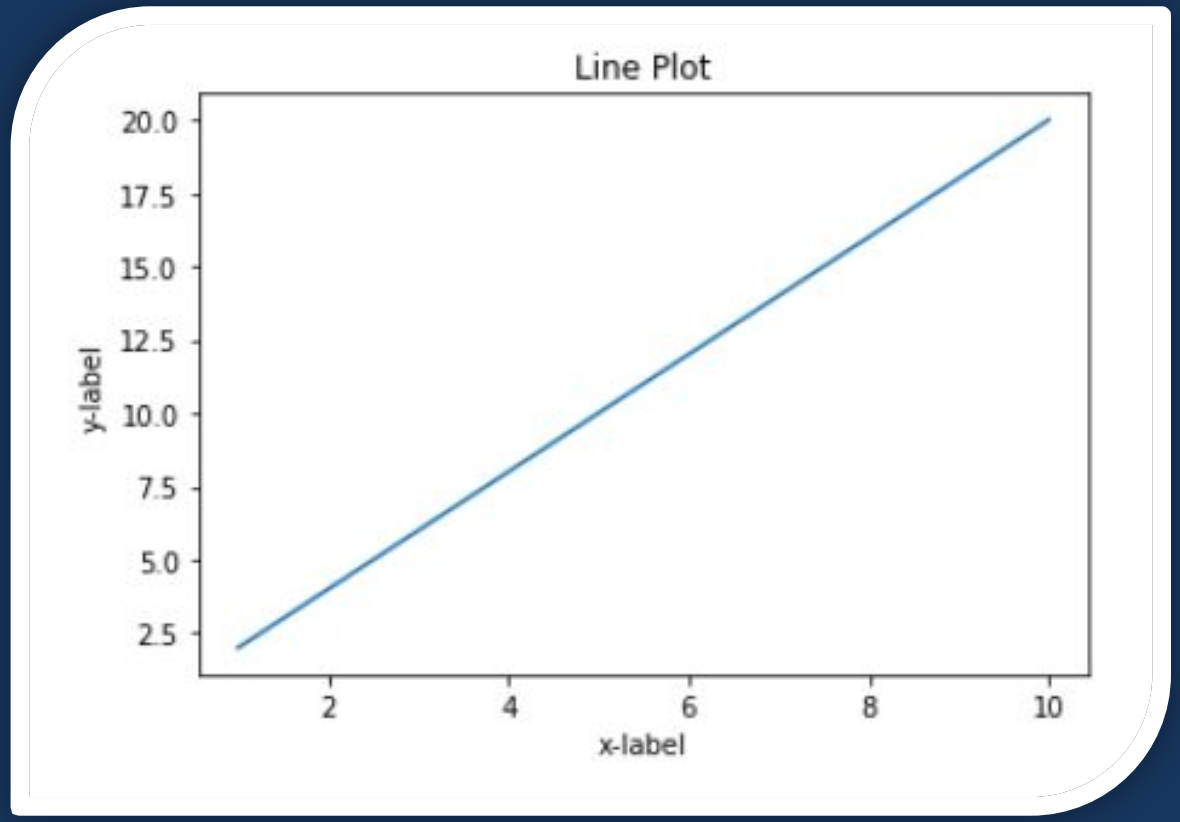
```
In [6]: plt.plot(x,y)
        plt.show()
```



Line Plot

Adding Title and Labels

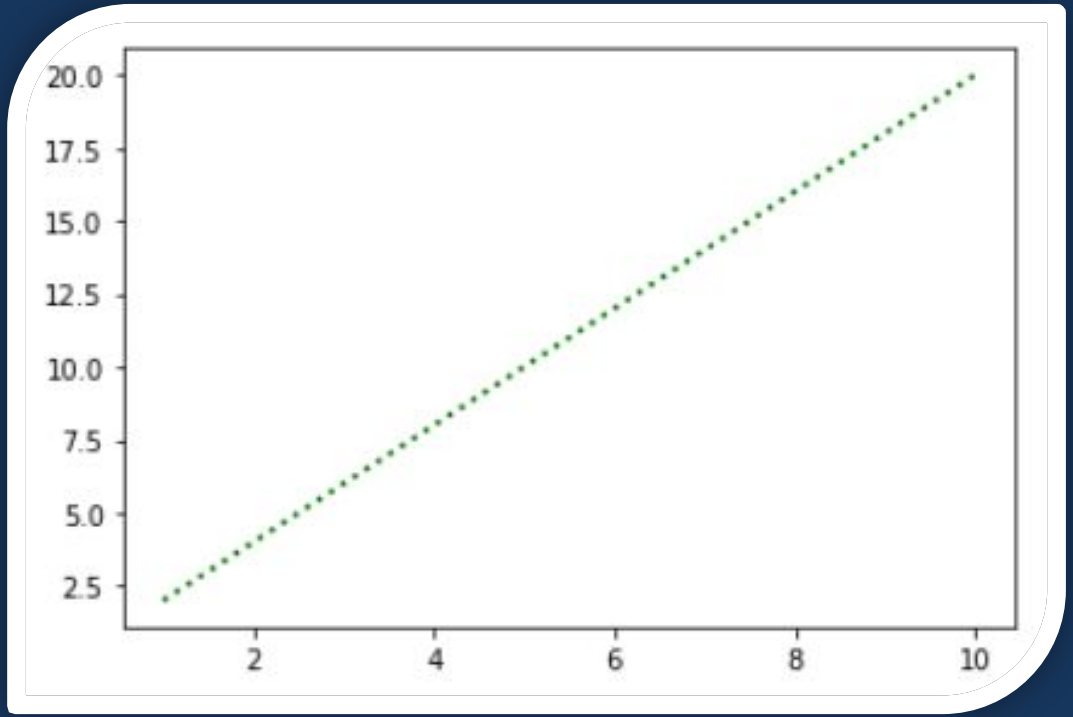
```
In [8]: plt.plot(x,y)
        plt.title("Line Plot")
        plt.xlabel("x-label")
        plt.ylabel("y-label")
        plt.show()
```



Line Plot

Changing Line Aesthetics

```
In [10]: plt.plot(x,y,color='g',linestyle=':',linewidth=2)
         plt.show()
```



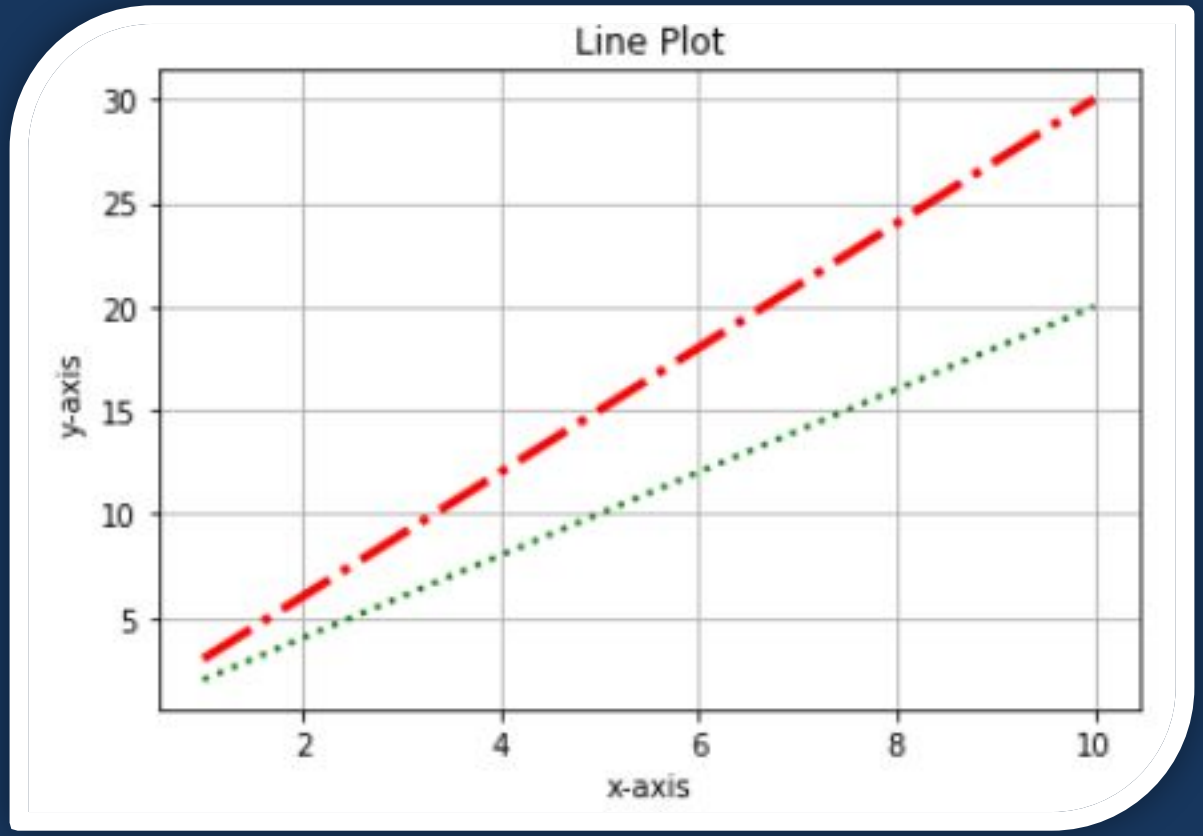
Line Plot

Adding two lines in the same plot

```
In [2]: x=np.arange(1,11)
        y1=2*x
        y2=3*x
```

```
In [11]: plt.plot(x,y1,color='g',linestyle=':',linewidth=2)
         plt.plot(x,y2,color='r',linestyle='-.',linewidth=3)
         plt.title("Line Plot")
         plt.xlabel("x-axis")
         plt.ylabel("y-axis")
         plt.grid(True)
         plt.show()
```

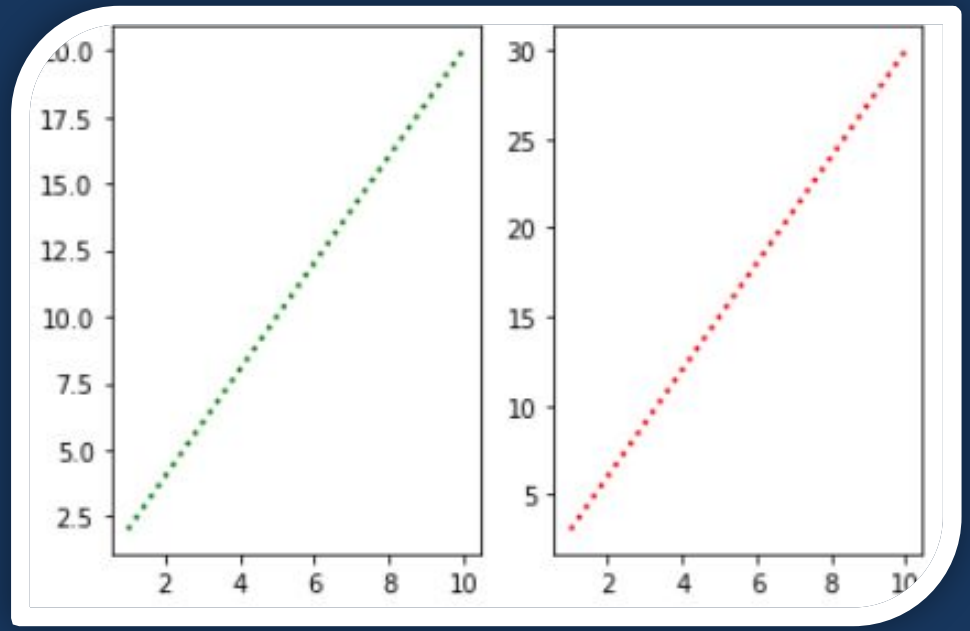
Line Plot



Line Plot

Adding sub-plots

```
x=np.arange(1,11)
y1=2*x
y2=3*x

plt.subplot(1,2,1)
plt.plot(x,y1,color='g',linestyle=':',linewidth=2)

plt.subplot(1,2,2)
plt.plot(x,y2,color='r',linestyle=':',linewidth=2)

plt.show()
```

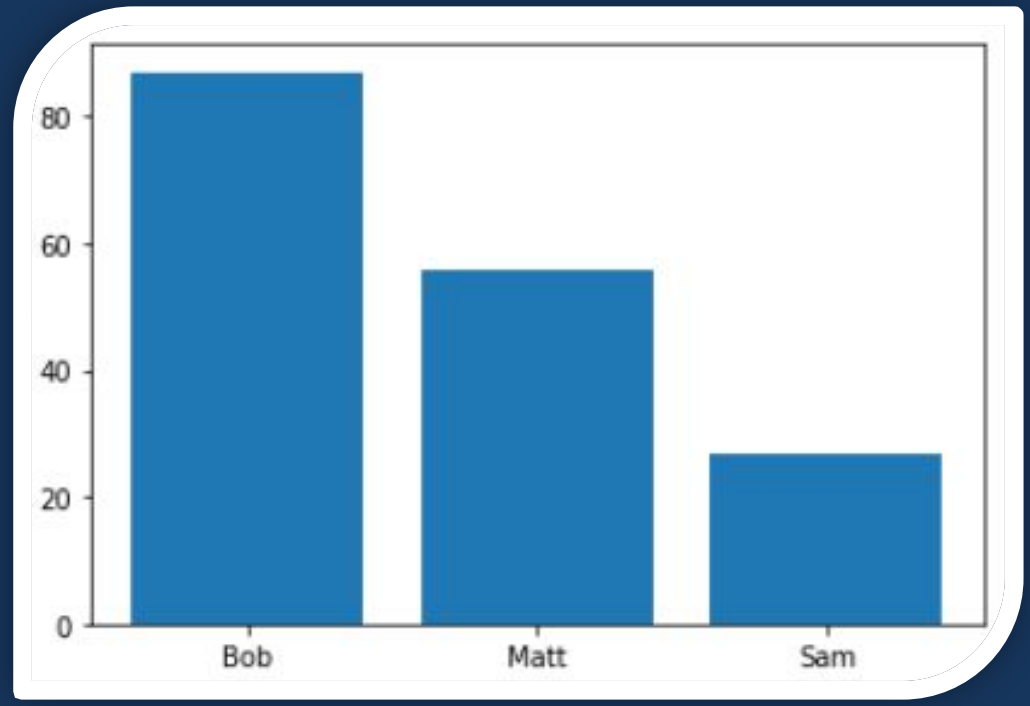


Bar Plot

```
[39]: student = {"Bob":87,"Matt":56,"Sam":27}
```

```
In [40]: names = list(student.keys())
         values = list(student.values())
```

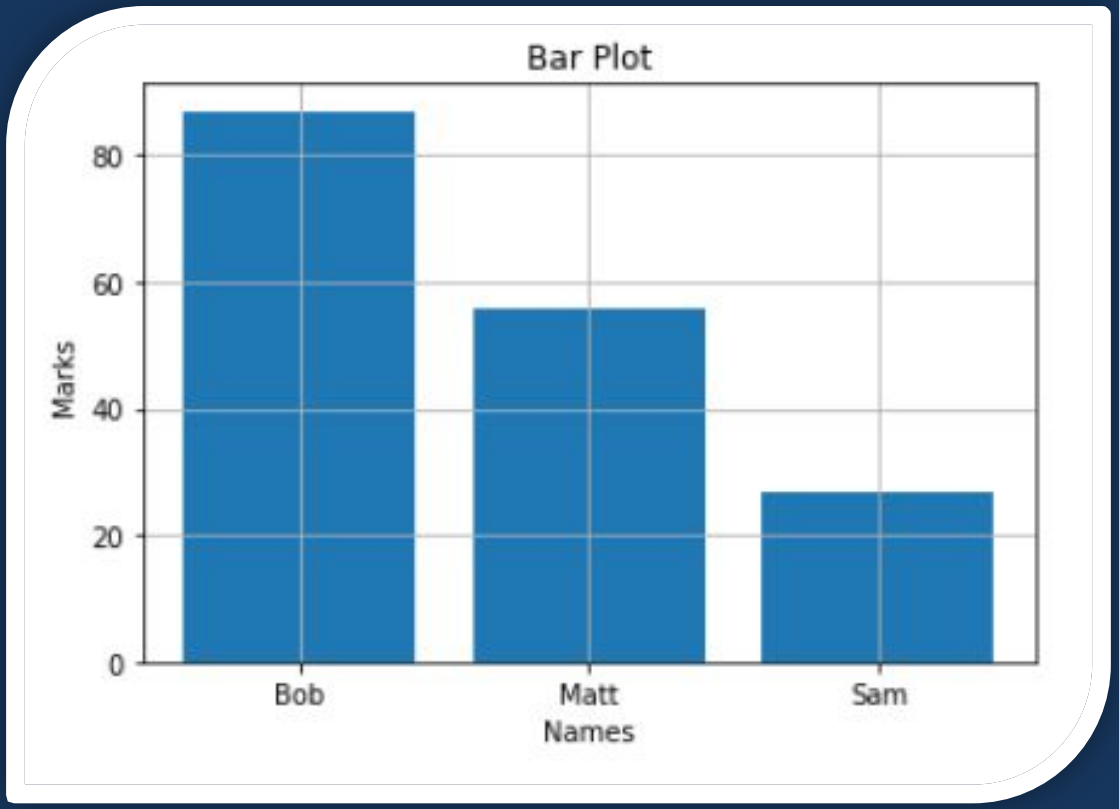
```
In [42]: plt.bar(names,values)
         plt.show()
```



Bar Plot

Adding Title and Labels

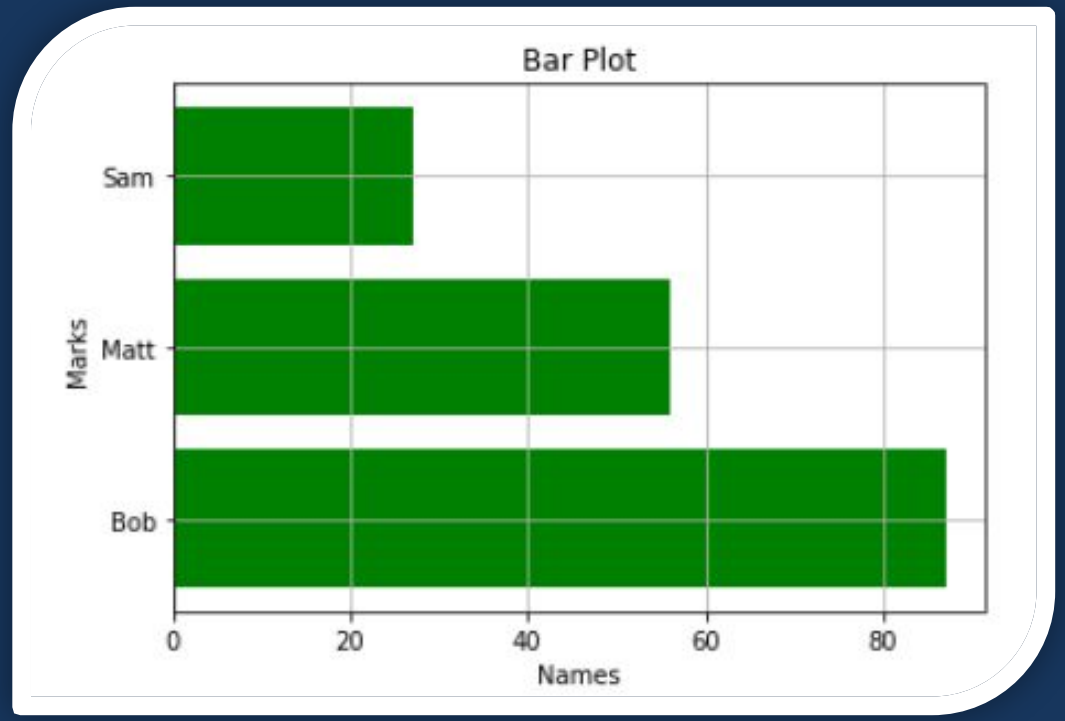
```
In [16]: plt.bar(names,values)
         plt.title("Bar Plot")
         plt.xlabel("Names")
         plt.ylabel("Marks")
         plt.grid(True)
         plt.show()
```



Horizontal Bar Plot

Horizontal Bar Plot

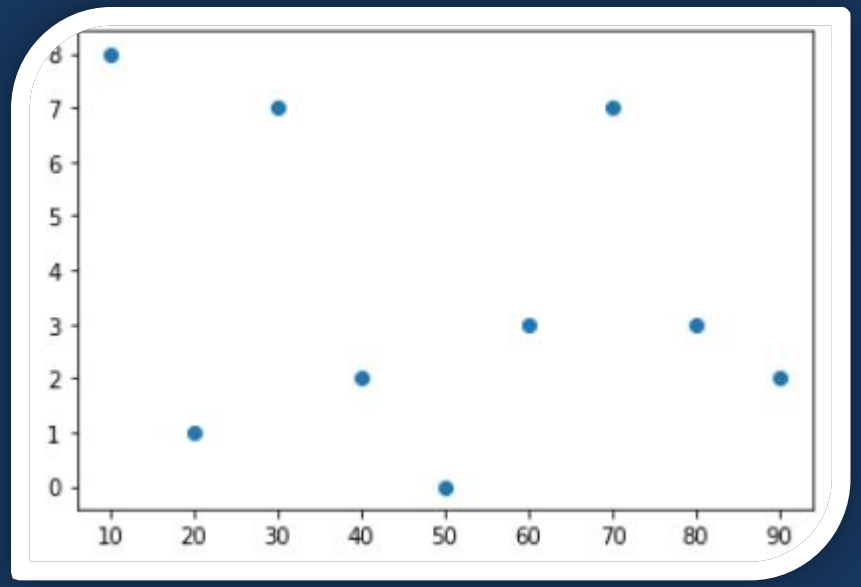
```
In [44]: plt.barh(names,values,color='g')
         plt.title("Bar Plot")
         plt.xlabel("Names")
         plt.ylabel("Marks")
         plt.grid(True)
         plt.show()
```



Scatter Plot

Creating a basic scatter-plot

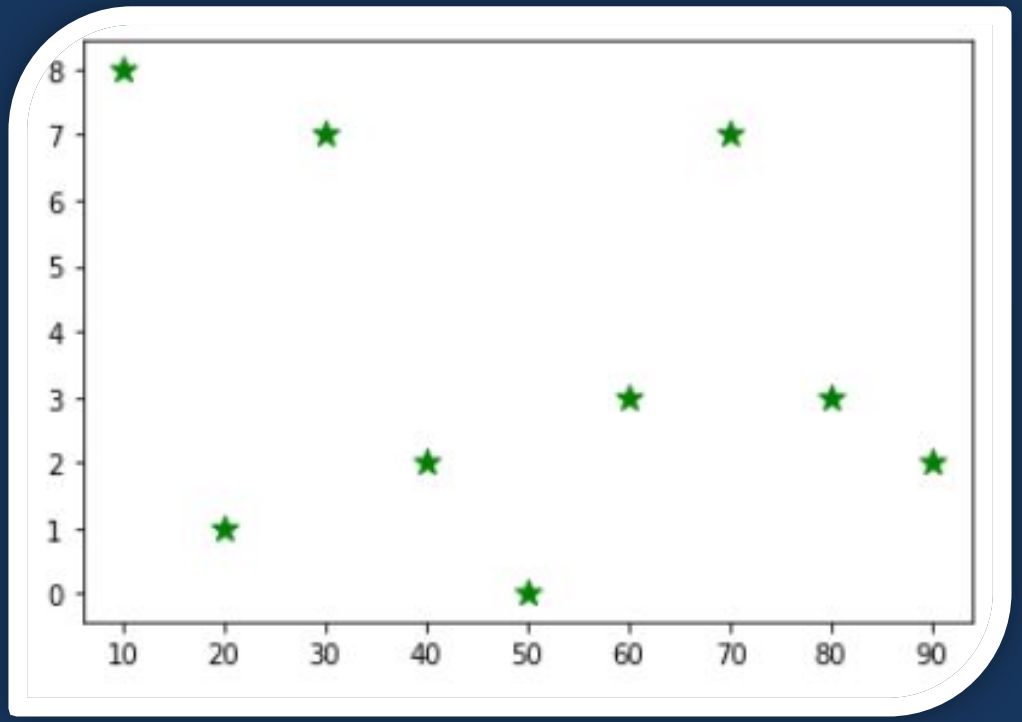
```
x=[10,20,30,40,50,60,70,80,90]
a=[8,1,7,2,0,3,7,3,2]

plt.scatter(x,a)
plt.show()
```



Scatter Plot

Changing Mark Aesthetics

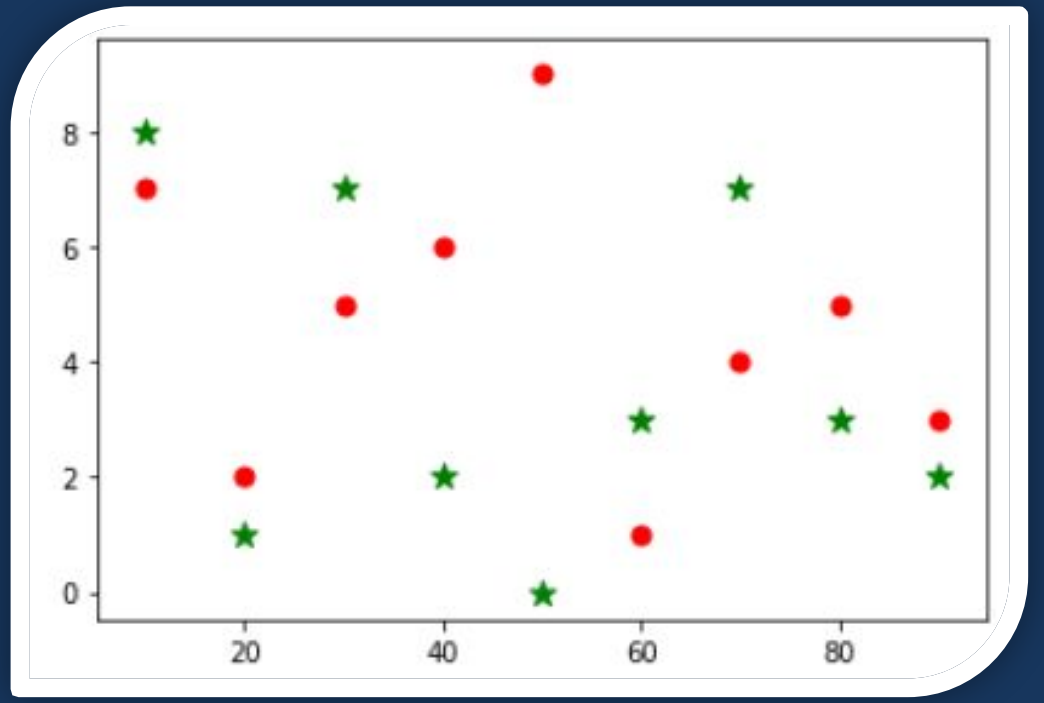
```
In [7]: x=[10,20,30,40,50,60,70,80,90]
        a=[8,1,7,2,0,3,7,3,2]
        plt.scatter(x,a,marker="*",c="g",s=100)
        plt.show()
```



Scatter Plot

Adding two markers in the same plot

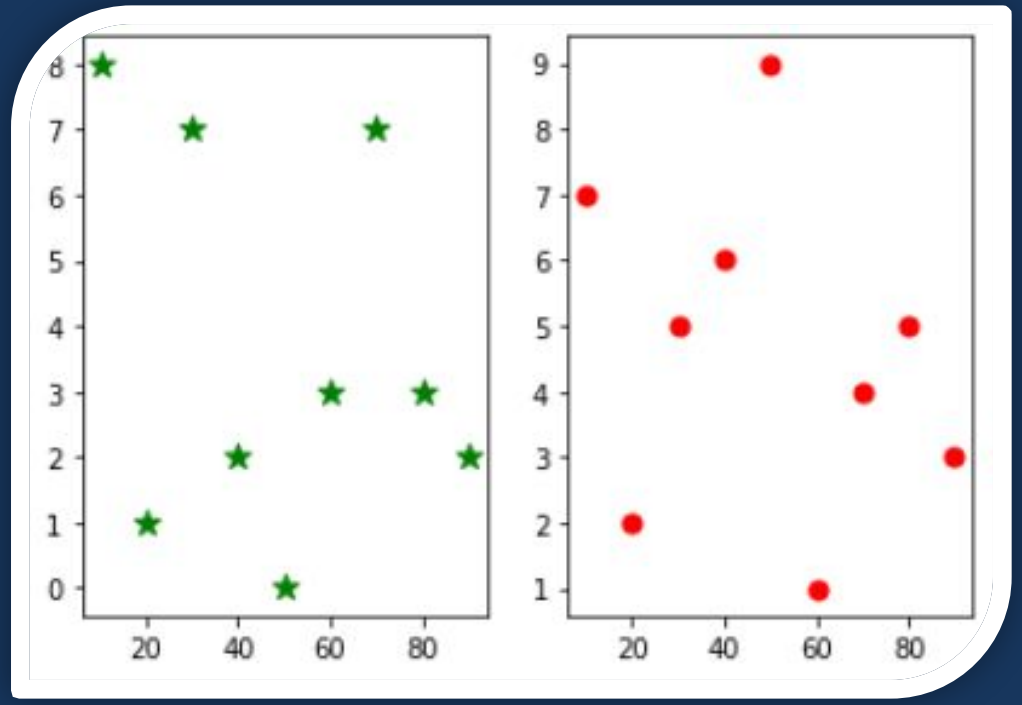
```
In [10]: x=[10,20,30,40,50,60,70,80,90]
         a=[8,1,7,2,0,3,7,3,2]
         b=[7,2,5,6,9,1,4,5,3]
         plt.scatter(x,a,marker="*",c="g",s=100)
         plt.scatter(x,b,marker=".",c="r",s=200)
         plt.show()
```



Scatter Plot

Adding sub-plots

```
x=[10,20,30,40,50,60,70,80,90]
a=[8,1,7,2,0,3,7,3,2]
b=[7,2,5,6,9,1,4,5,3]

plt.subplot(1,2,1)
plt.scatter(x,a,marker="*",c="g",s=100)

plt.subplot(1,2,2)
plt.scatter(x,b,marker=".",c="r",s=200)
plt.show()
```



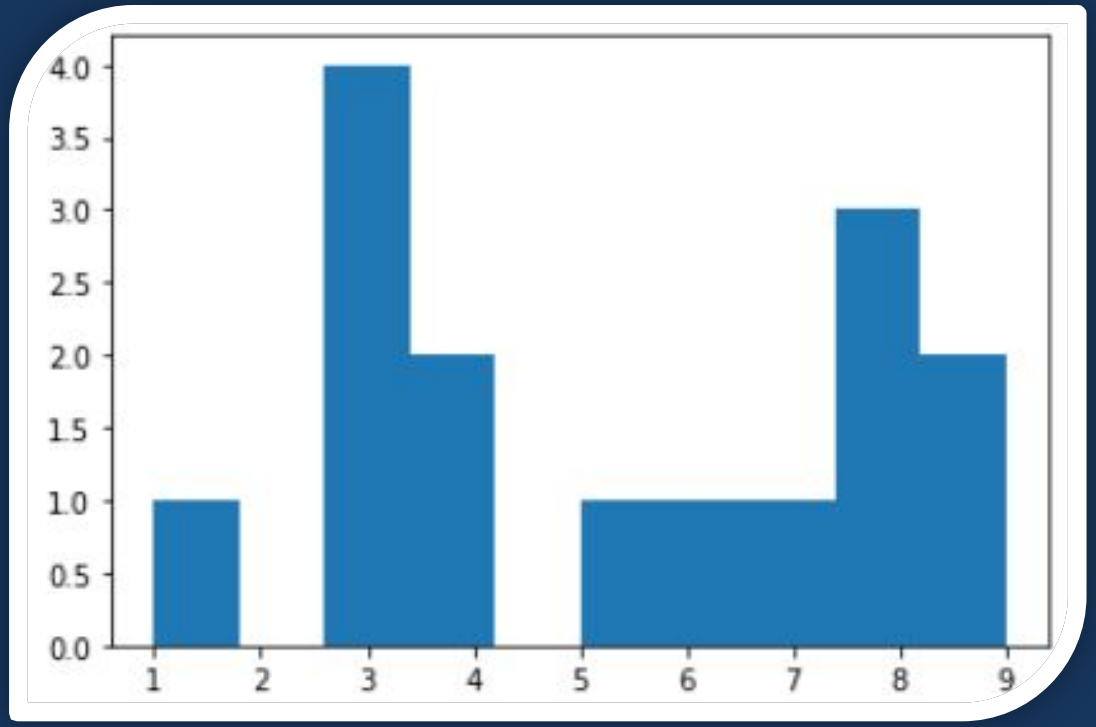
Histogram

Creating data

```
data = [1,3,3,3,3,9,9,5,4,4,8,8,8,6,7]
```

Making Histogram

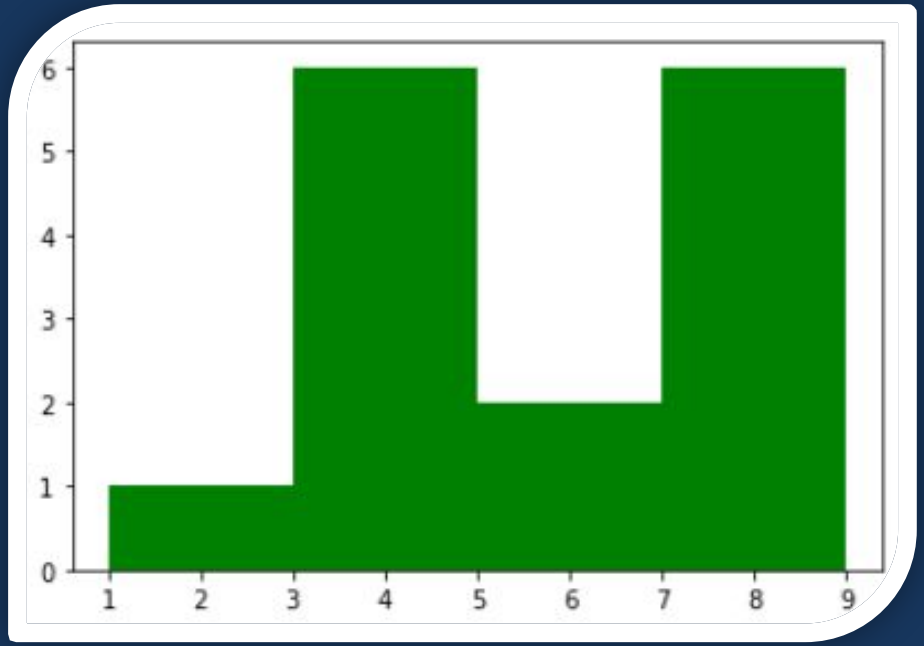
```
plt.hist(data)
plt.show()
```



Histogram

Changing Aesthetics

```
In [24]: plt.hist(data,color="g",bins=4)
         plt.show()
```

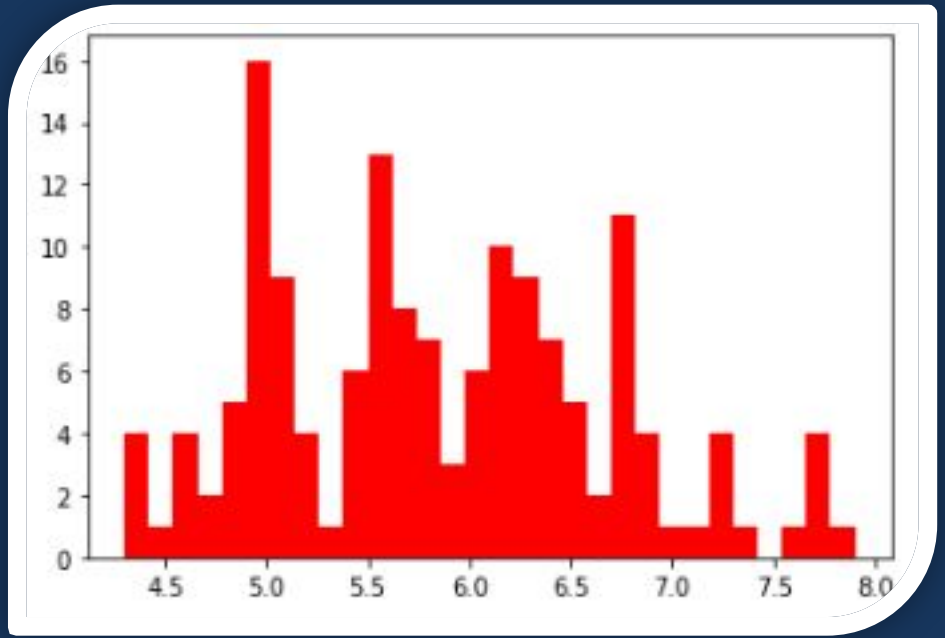


Histogram

Working with a dataset

```
iris=pd.read_csv('iris.csv')
iris.head()
```

```
plt.hist(iris['Sepal.Length'],bins=30,color="r")
plt.show()
```



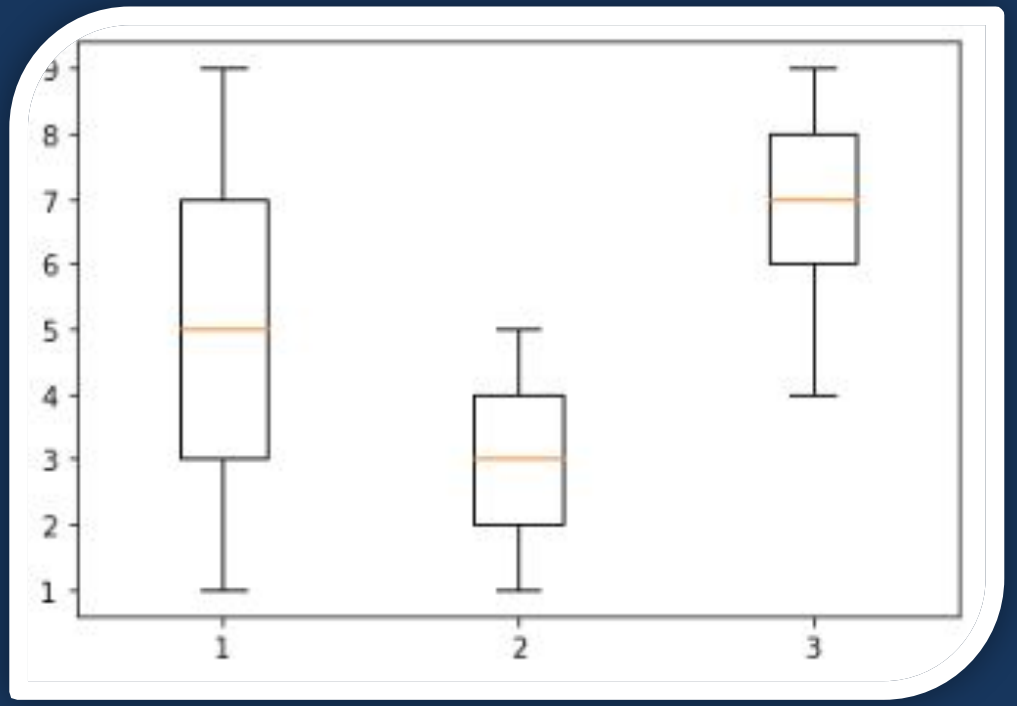
Box-Plot

Creating data

```
one = [1,2,3,4,5,6,7,8,9]
two = [1,2,3,4,5,4,3,2,1]
three = [6,7,8,9,8,7,6,5,4]

data =  list([one,two,three])
```

Making Plot

```
plt.boxplot(data)
plt.show()
```



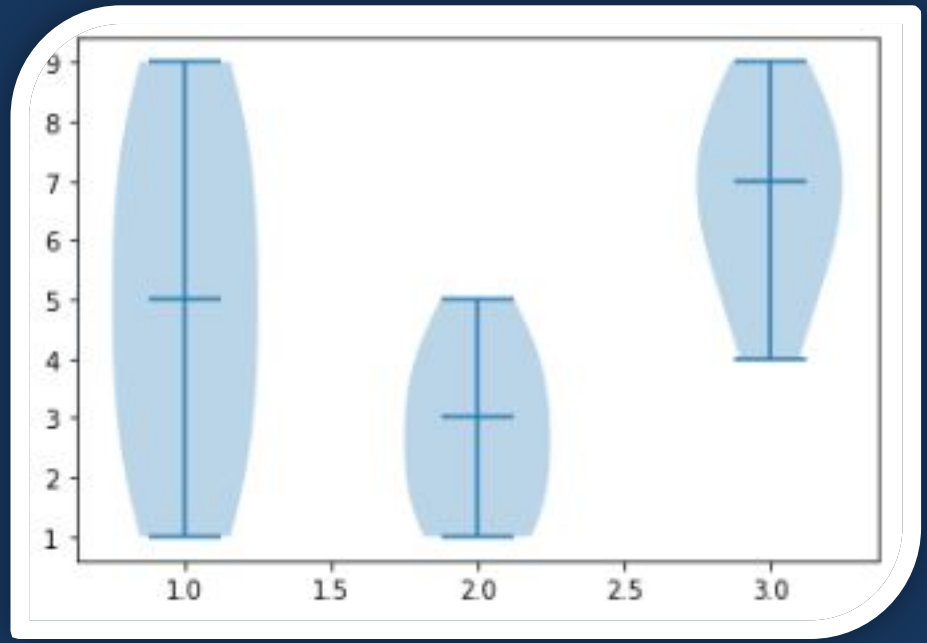
Violin-Plot

Creating data

```
one = [1,2,3,4,5,6,7,8,9]
two = [1,2,3,4,5,4,3,2,1]
three = [6,7,8,9,8,7,6,5,4]

data =  list([one,two,three])
```

Making Plot

```
plt.violinplot(data,showmedians=True)
plt.show()
```



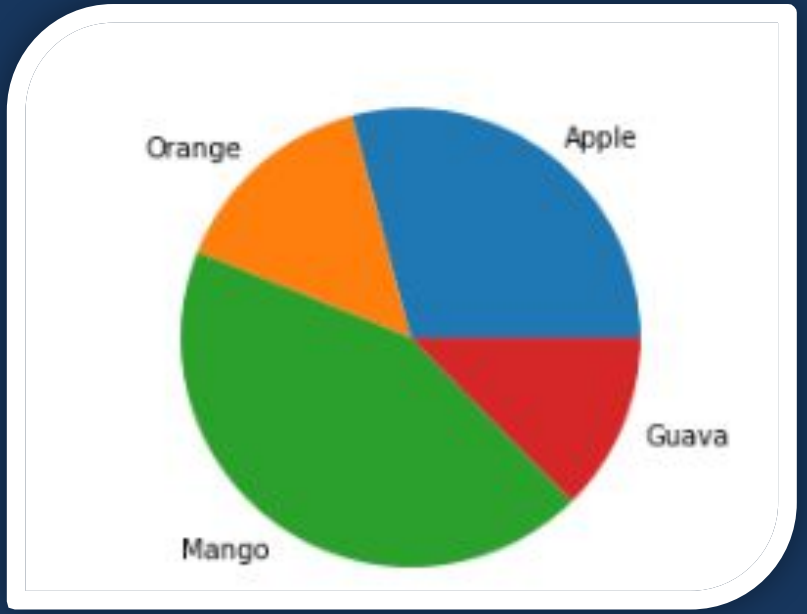
Pie-Chart

Creating data

```
fruit = ['Apple','Orange','Mango','Guava']
quantity = [67,34,100,29]
```

Making Plot

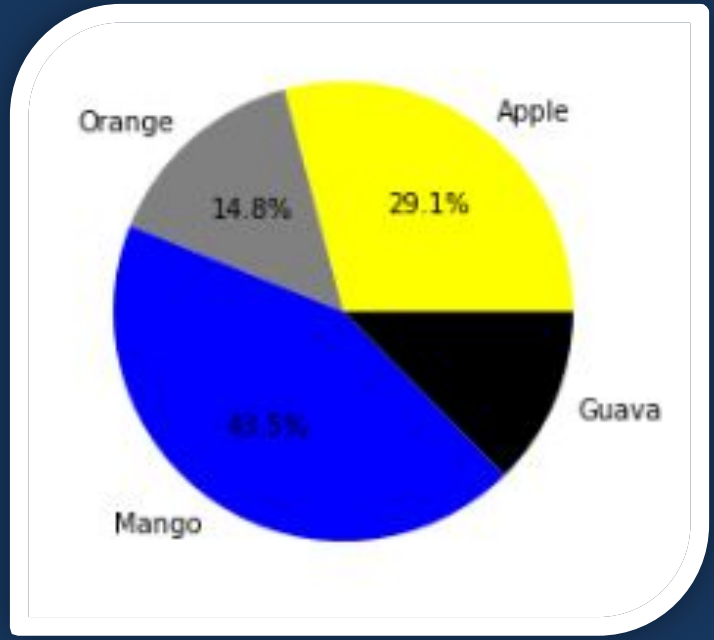
```
plt.pie(quantity,labels=fruit)
plt.show()
```



Pie-Chart

Changing Aesthetics

```
plt.pie(quantity,labels=fruit,autopct='%0.1f%%'
        ,colors=['yellow','grey','blue','black'])
plt.show()
```



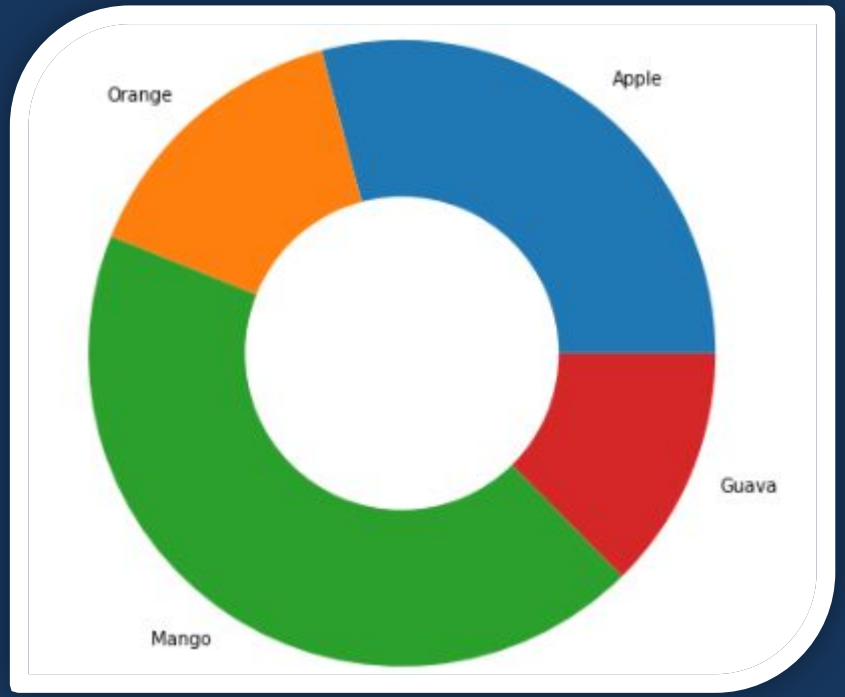
DoughNut-Chart

Creating Data

```
fruit = ['Apple','Orange','Mango','Guava']
quantity = [67,34,100,29]
```

Making Plot

```
plt.pie(quantity,labels=fruit,radius=2)
plt.pie([1],colors=['w'],radius=1)
plt.show()
```



SeaBorn Line Plot

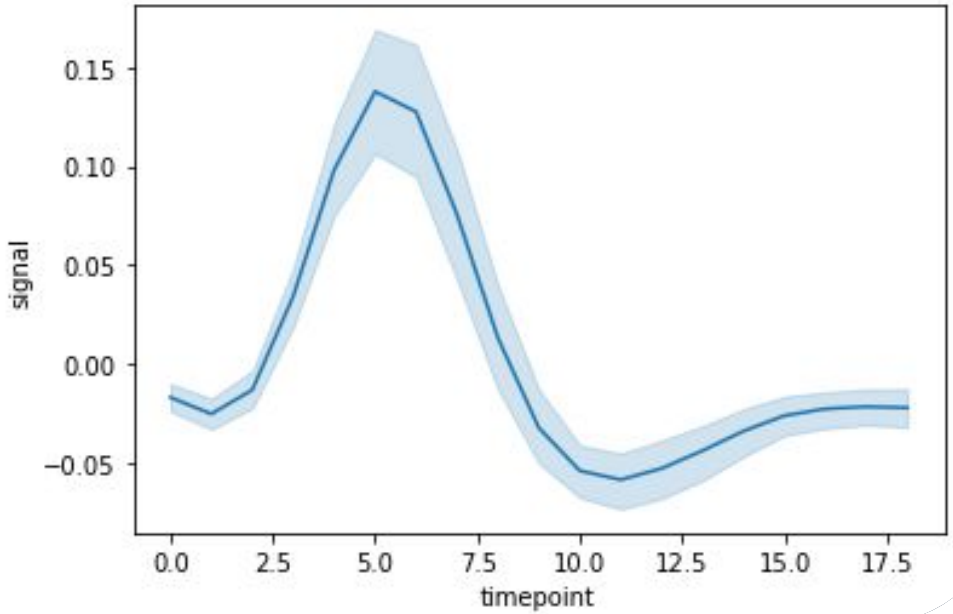
```
In [10]: import seaborn as sns
         from matplotlib import pyplot as plt
```

```
In [18]: fmri = sns.load_dataset("fmri")
         fmri.head()
```

Out[18]:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

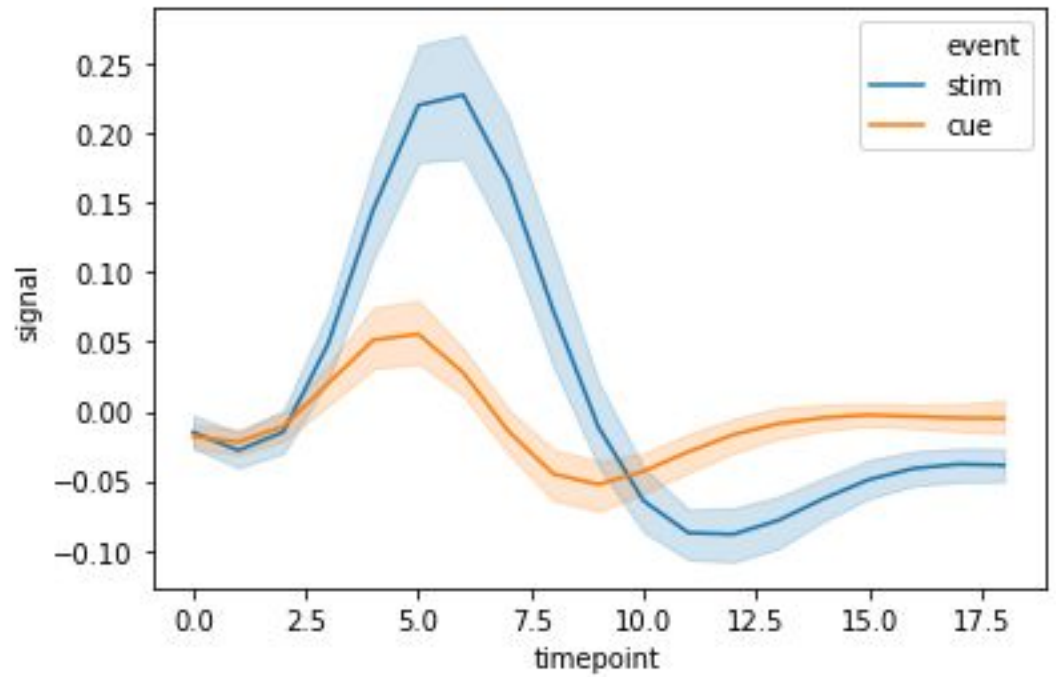
```
In [20]: sns.lineplot(x="timepoint", y="signal", data=fmri)
         plt.show()
```



SeaBorn Line Plot

Grouping data with ‘hue’

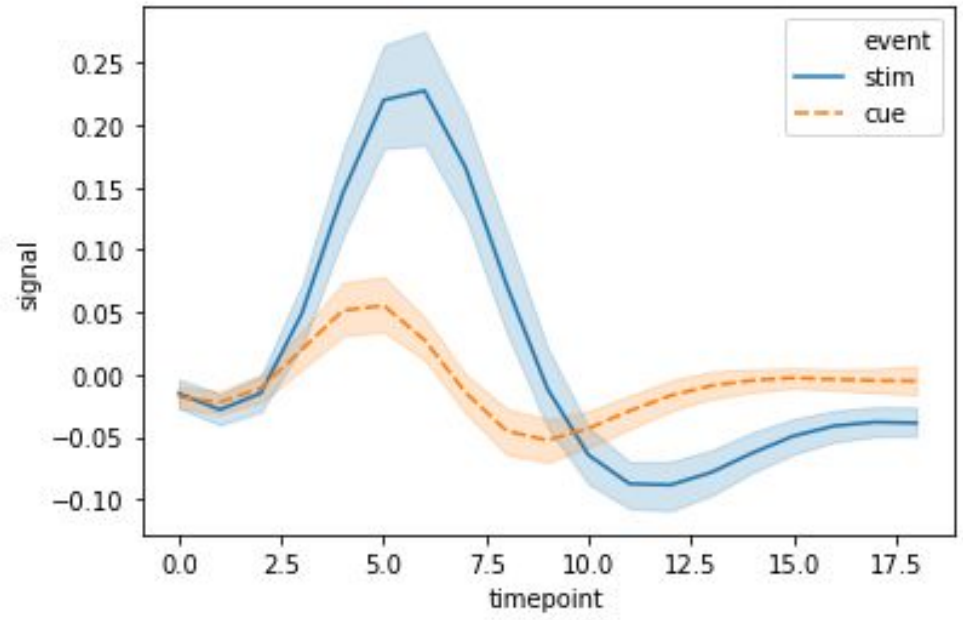
```
In [21]: sns.lineplot(x="timepoint", y="signal", data=fmri,hue="event")
         plt.show()
```



SeaBorn Line Plot

Adding Styles

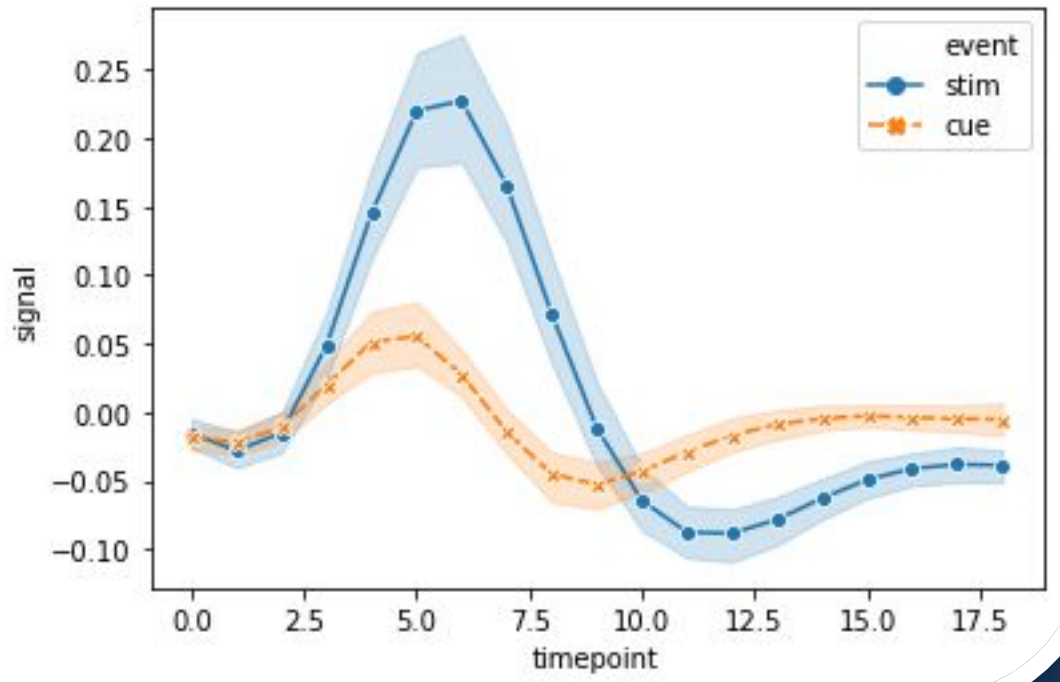
```
In [22]: sns.lineplot(x="timepoint", y="signal", data=fmri,hue="event",style="event")
         plt.show()
```



SeaBorn Line Plot

Adding Markers

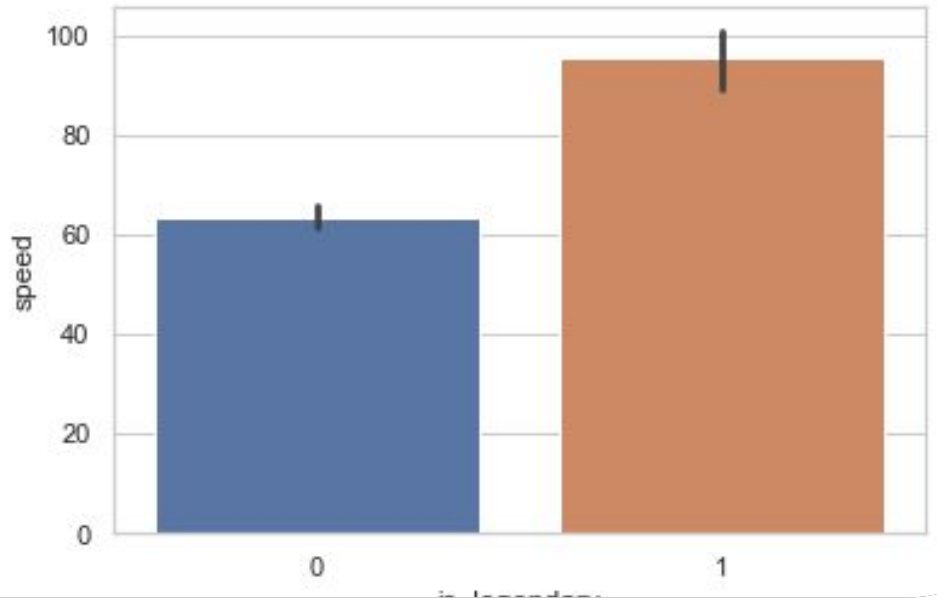
```
[24]: sns.lineplot(x="timepoint", y="signal",
                   hue="event", style="event",
                   markers=True, data=fmri)
plt.show()
```



SeaBorn Bar Plot

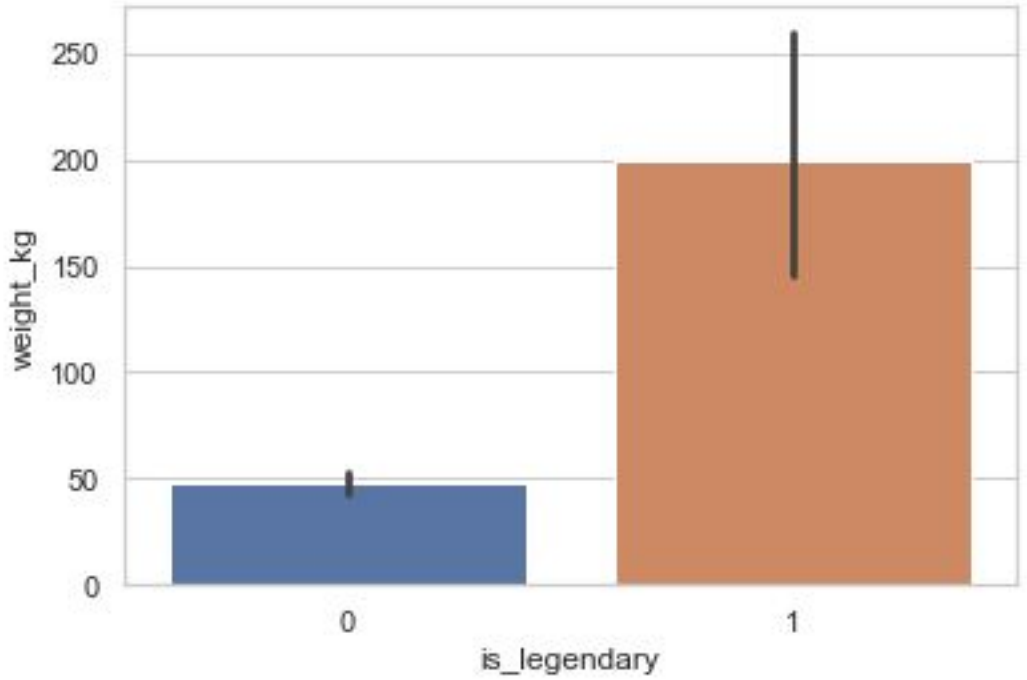
```
In [29]: import pandas as pd
         sns.set(style="whitegrid")
         pokemon=pd.read_csv('pokemon.csv')
```

```
In [31]: sns.barplot(x="is_legendary", y="speed", data=pokemon)
         plt.show()
```



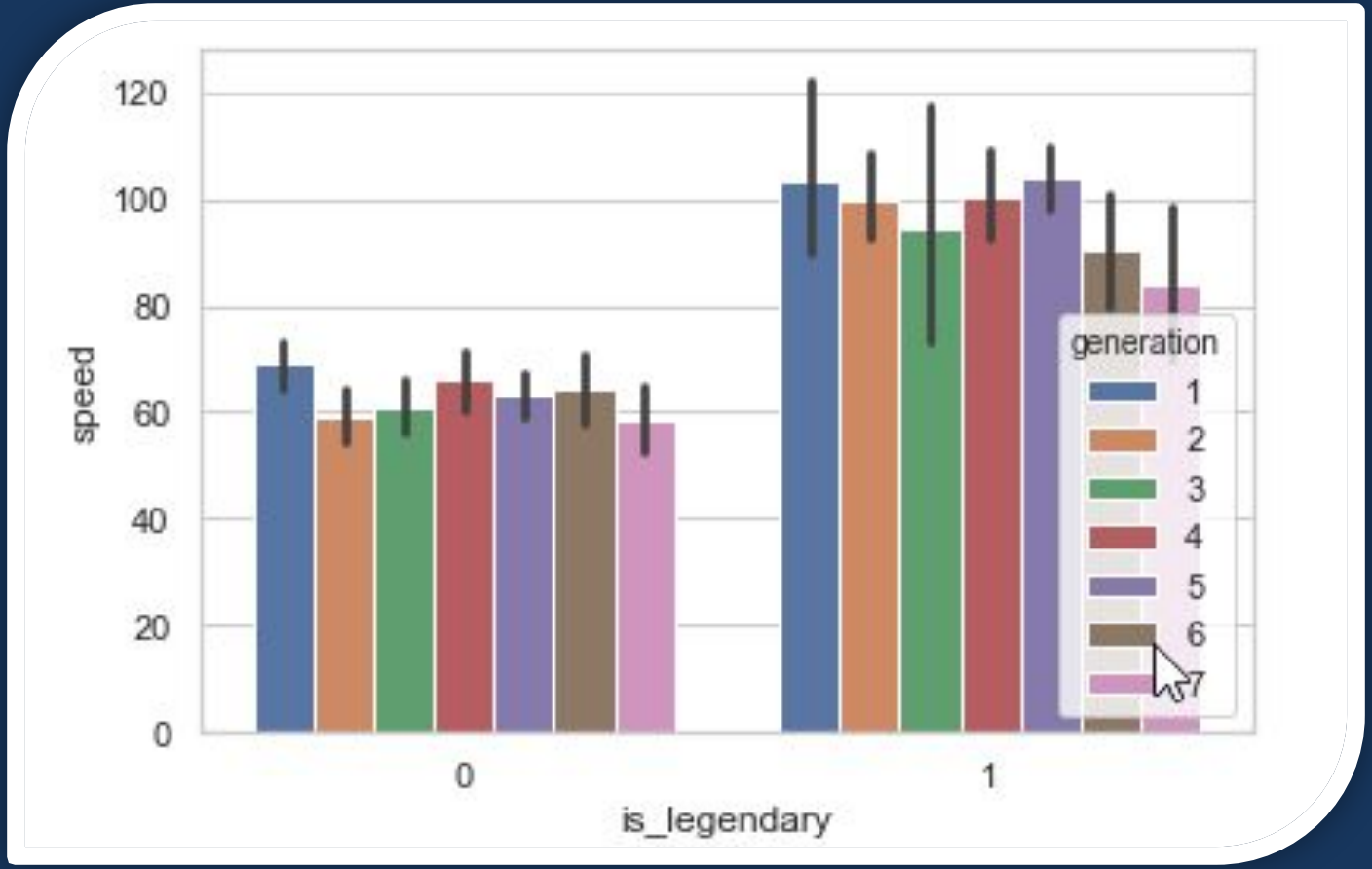
SeaBorn Bar Plot

```
In [43]: sns.barplot(x="is_legendary", y="weight_kg",data=pokemon)
         plt.show()
```



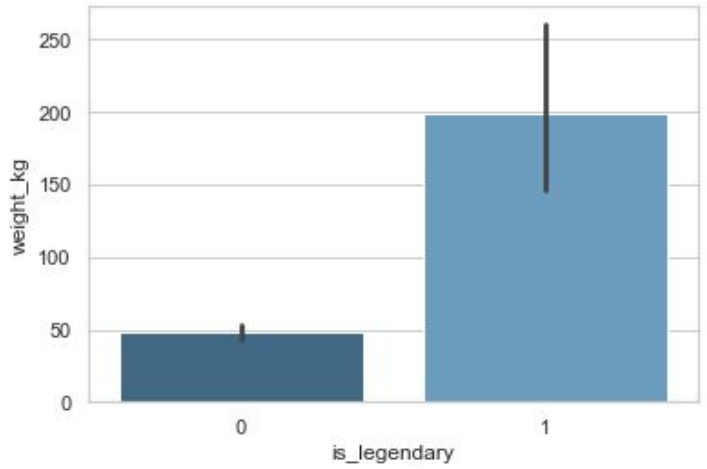
SeaBorn Bar Plot

```
In [32]: sns.barplot(x="is_legendary", y="speed", hue="generation",data=pokemon)
         plt.show()
```

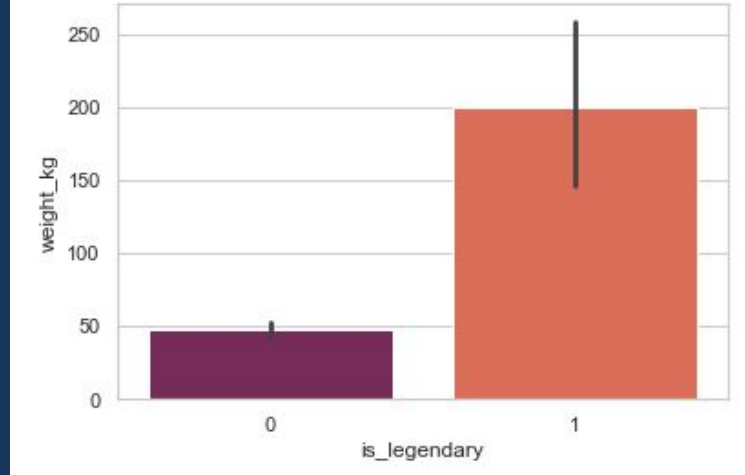


SeaBorn Bar Plot

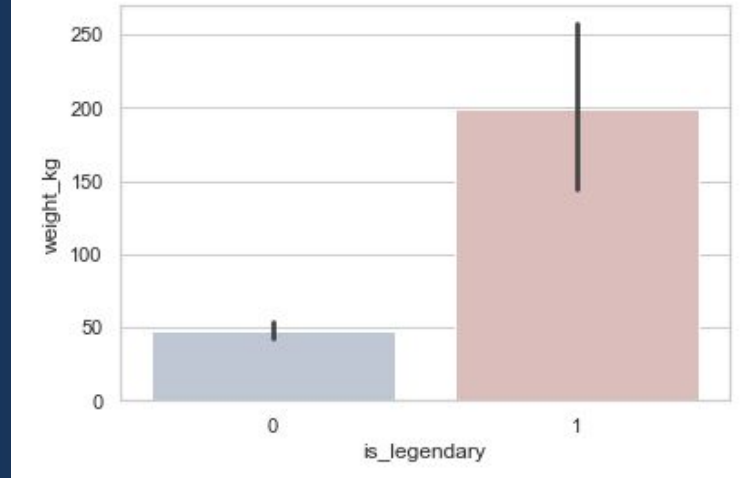
```
sns.barplot(x="is_legendary", y="weight_kg",
            data=pokemon,palette='Blues_d')
plt.show()
```




```
sns.barplot(x="is_legendary", y="weight_kg",
            data=pokemon,palette='rocket')
plt.show()
```

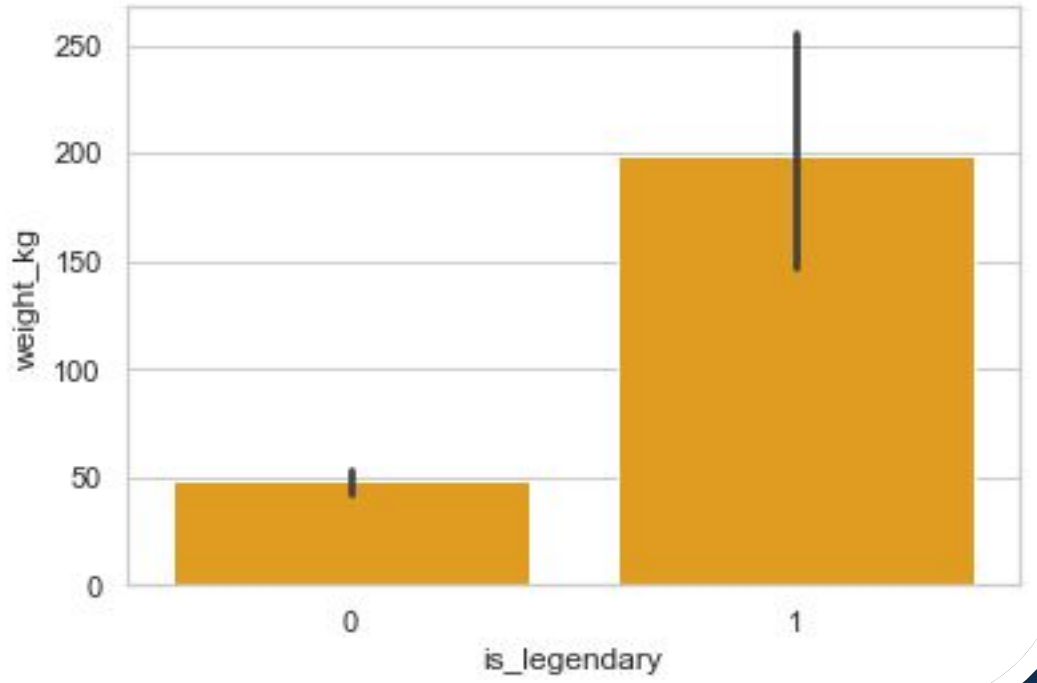



```
sns.barplot(x="is_legendary", y="weight_kg",
            data=pokemon,palette='vlag')
plt.show()
```



SeaBorn Bar Plot

```
In [50]: sns.barplot(x="is_legendary", y="weight_kg",
                     data=pokemon,color="orange")
         plt.show()
```

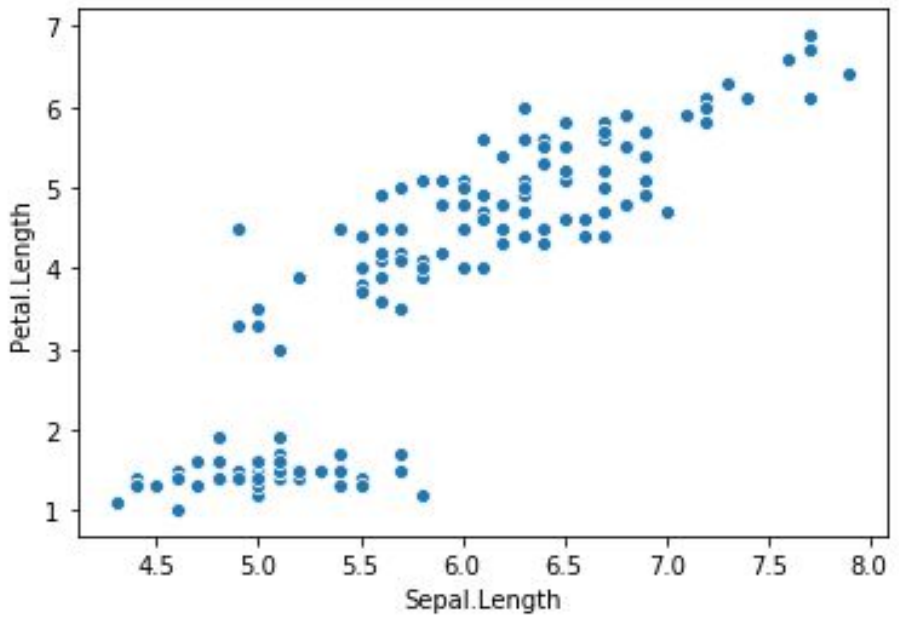


SeaBorn Scatterplot

```
In [5]: iris = pd.read_csv('iris.csv')
        iris.head()
```

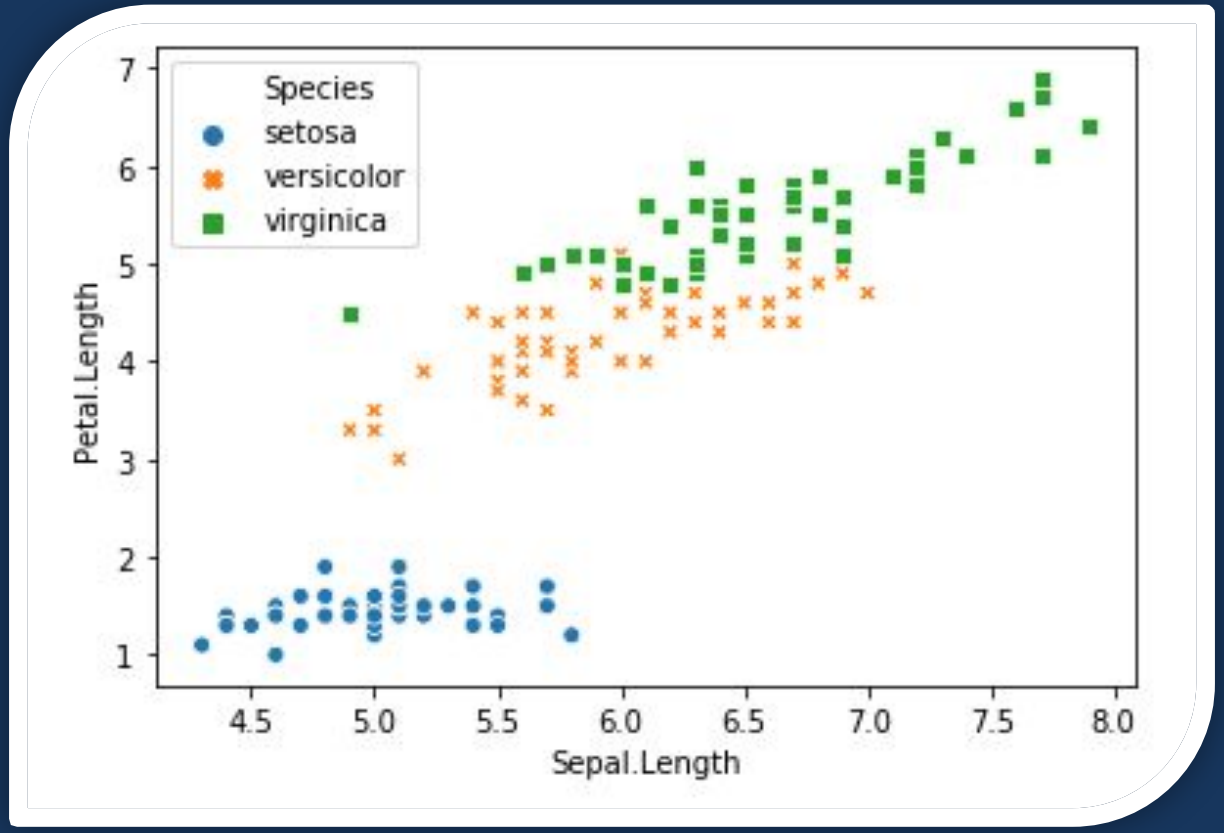
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
sns.scatterplot(x="Sepal.Length", y="Petal.Length", data=iris)
plt.show()
```



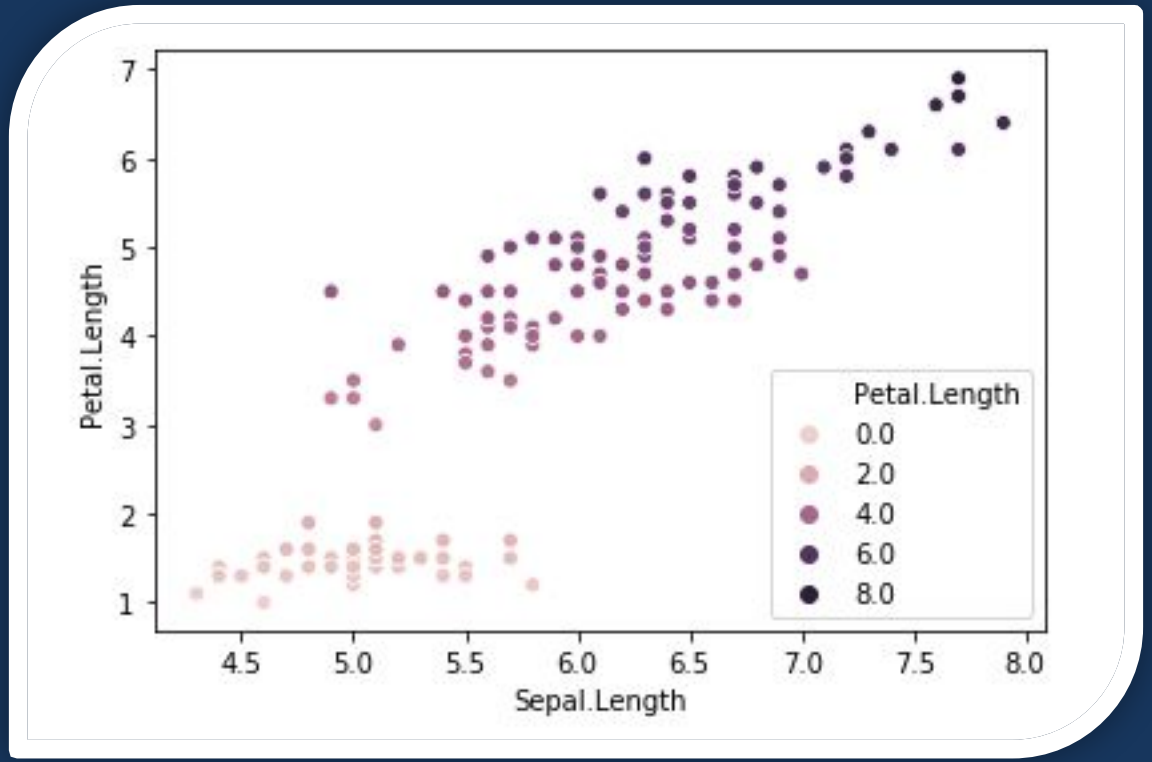
SeaBorn Scatterplot

```
sns.scatterplot(x="Sepal.Length", y="Petal.Length", data=iris,hue="Species",style="Species")
plt.show()
```



SeaBorn Scatterplot

```
sns.scatterplot(x='Sepal.Length',y='Petal.Length',data=iris,hue='Petal.Length')
```

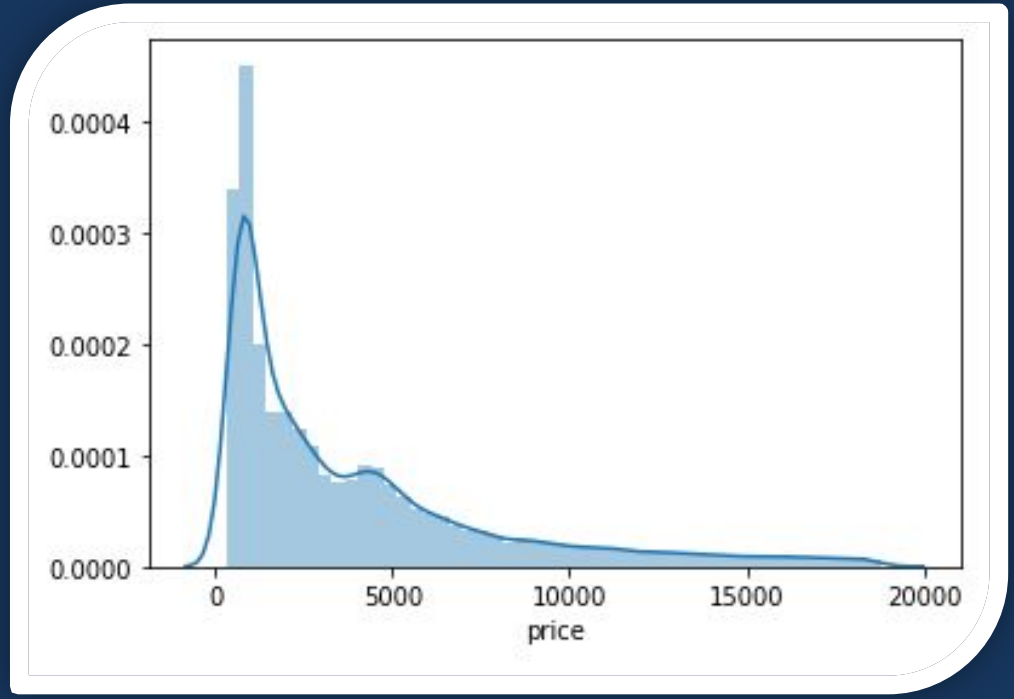


SeaBorn Histogram/Distplot

```
diamonds = pd.read_csv('diamonds.csv')
diamonds.head()
```

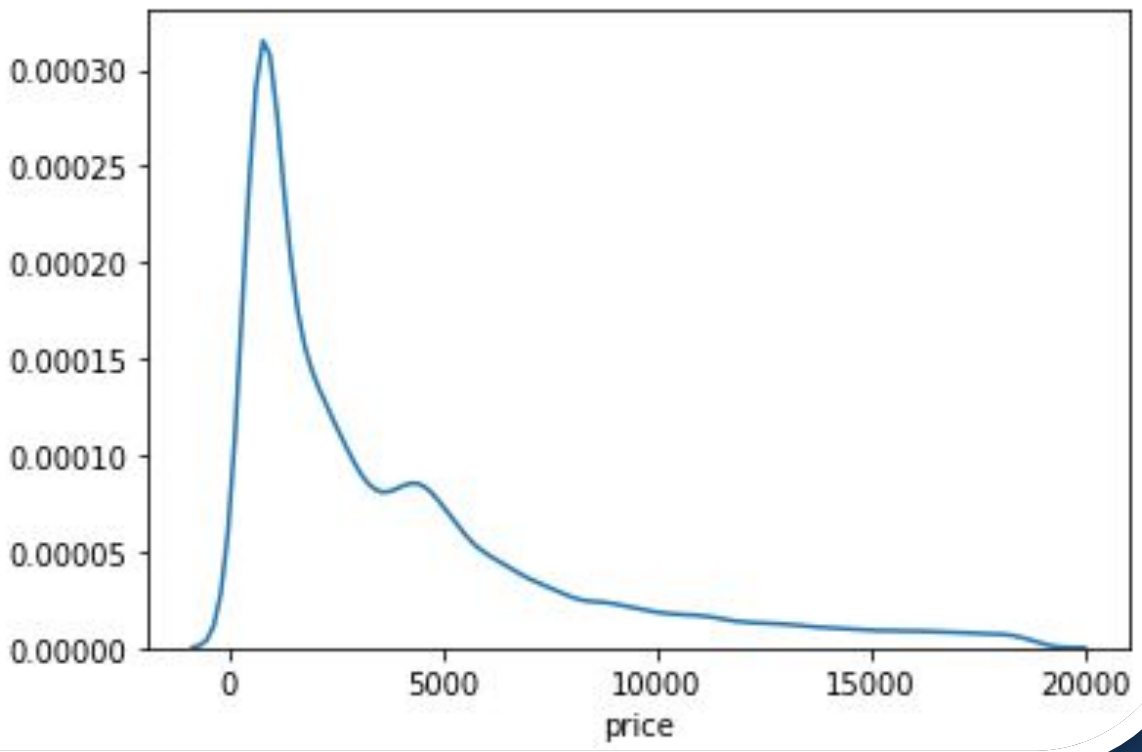
	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
sns.distplot(diamonds['price'])
plt.show()
```



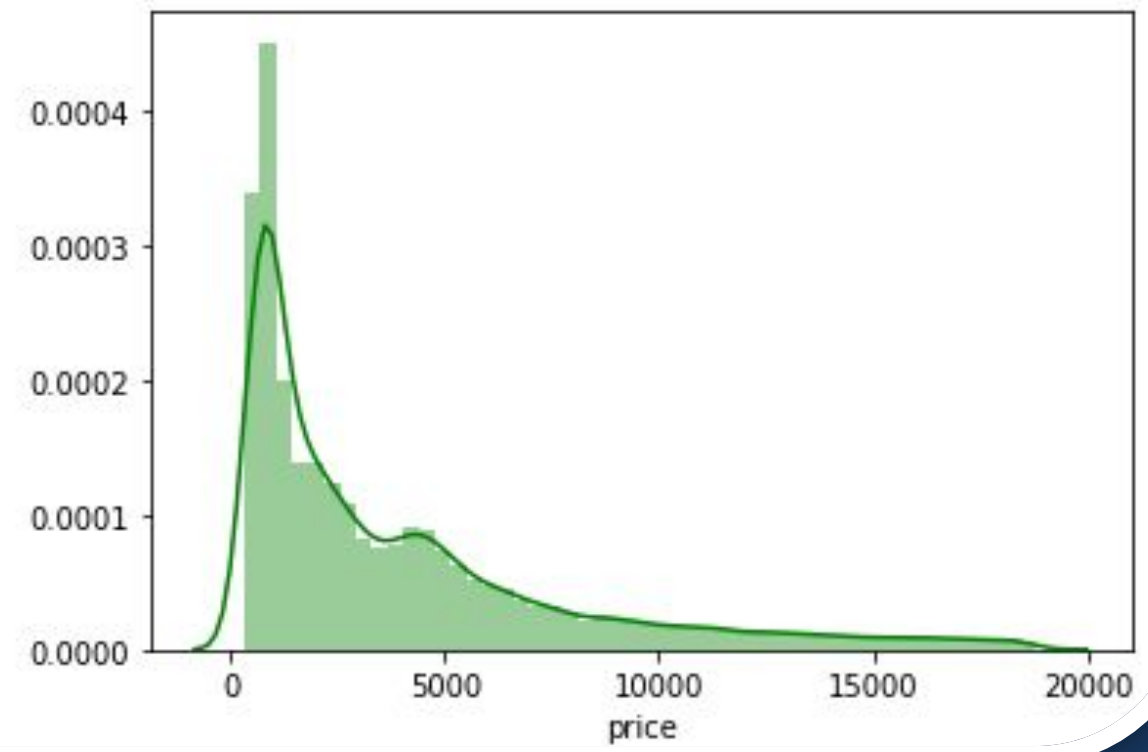
SeaBorn Histogram/Distplot

```
sns.distplot(diamonds['price'],hist=False)
plt.show()
```



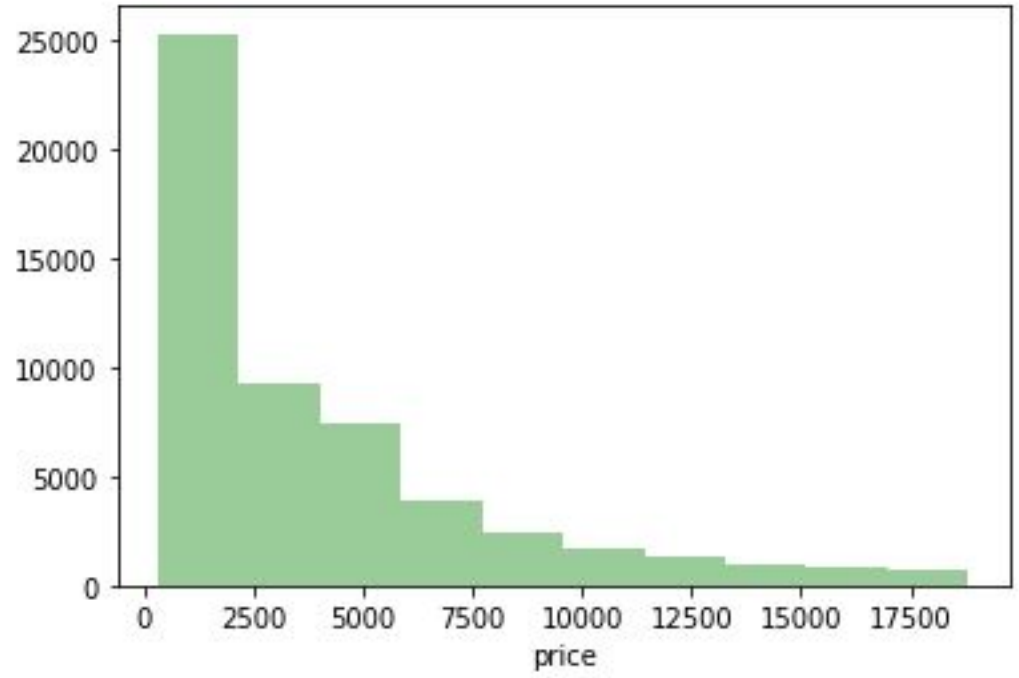
SeaBorn Histogram/Distplot

```
sns.distplot(diamonds['price'],color="green")
plt.show()
```



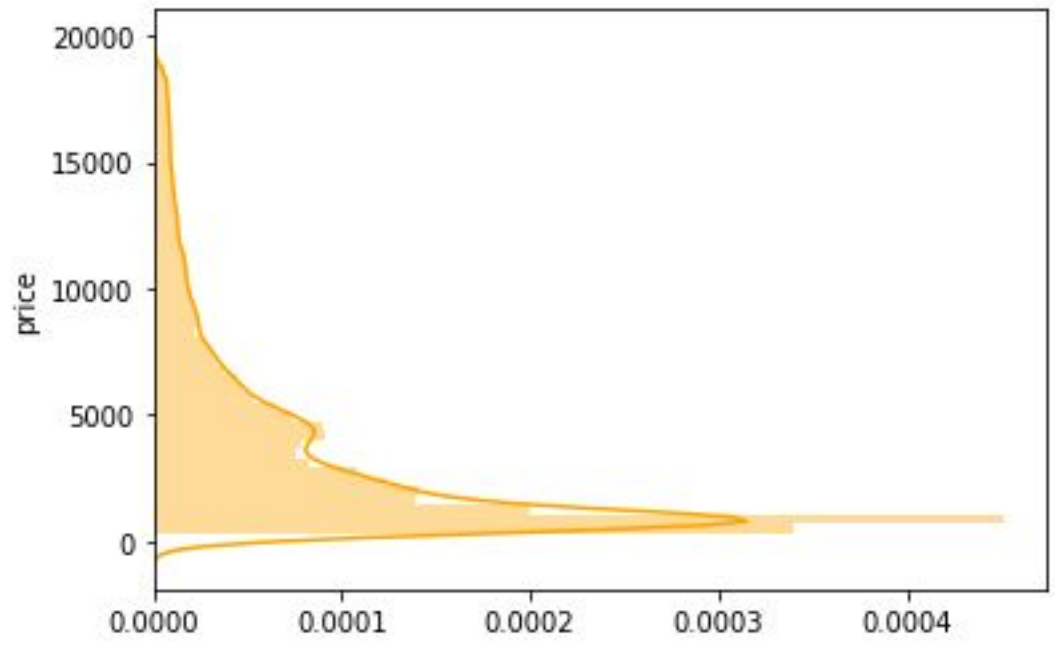
SeaBorn Histogram/Distplot

```
sns.distplot(diamonds['price'],color="green",bins=10,kde=False)
plt.show()
```



SeaBorn Histogram/Distplot

```
sns.distplot(diamonds['price'],color="orange",vertical=True)
plt.show()
```

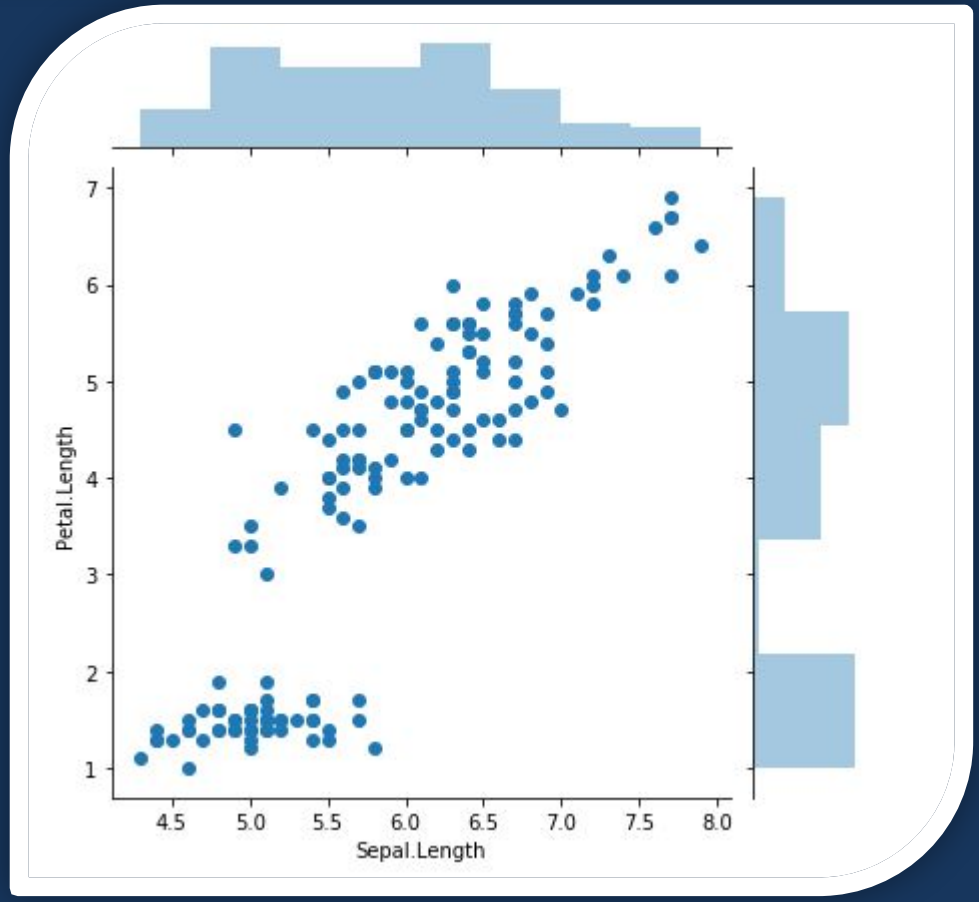


SeaBorn JointPlot

```
In [5]: iris = pd.read_csv('iris.csv')
        iris.head()
```

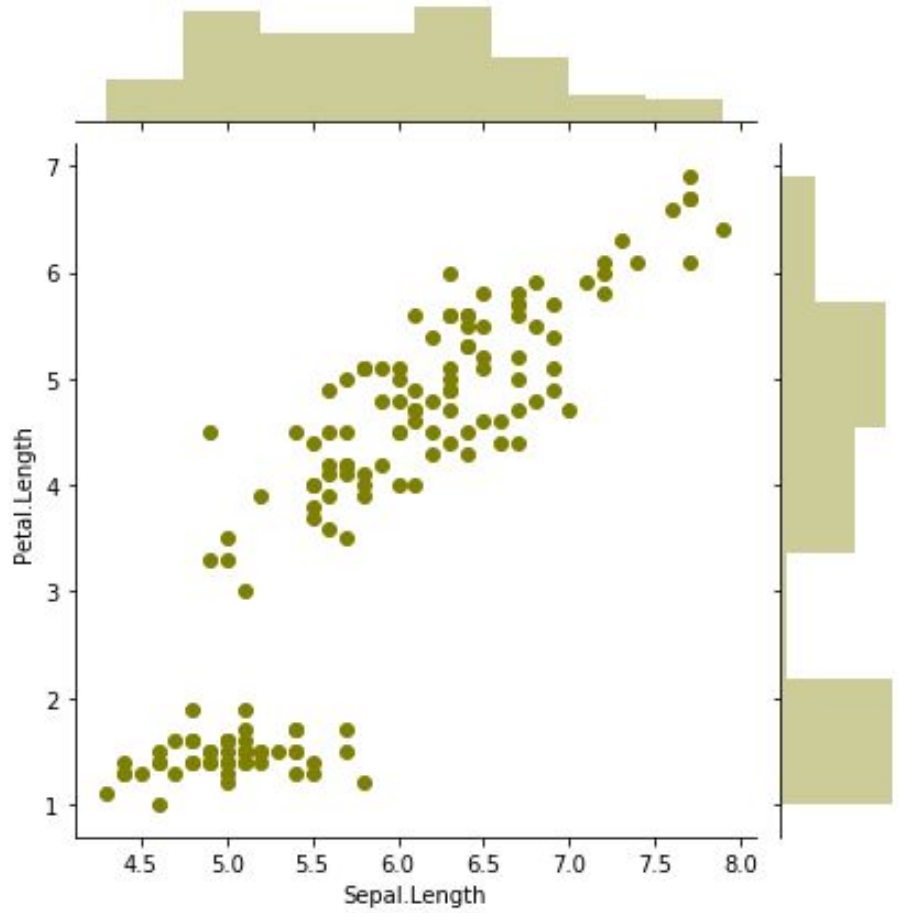
	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

```
sns.jointplot(x='Sepal.Length',y='Petal.Length',data=iris)
plt.show()
```



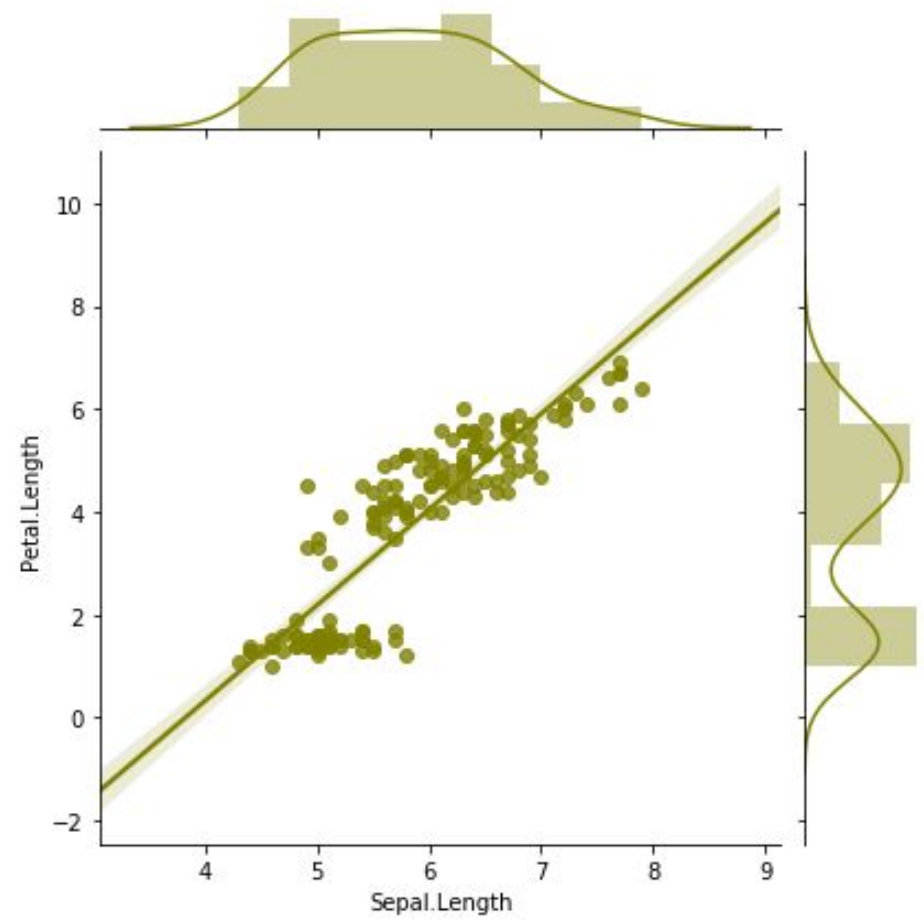
SeaBorn JointPlot

```
[62]: sns.jointplot(x='Sepal.Length',y='Petal.Length',data=iris,color="olive")
plt.show()
```



SeaBorn JointPlot

```
ns.jointplot(x='Sepal.Length',y='Petal.Length',data=iris,color="olive",kind="reg")
plt.show()
```

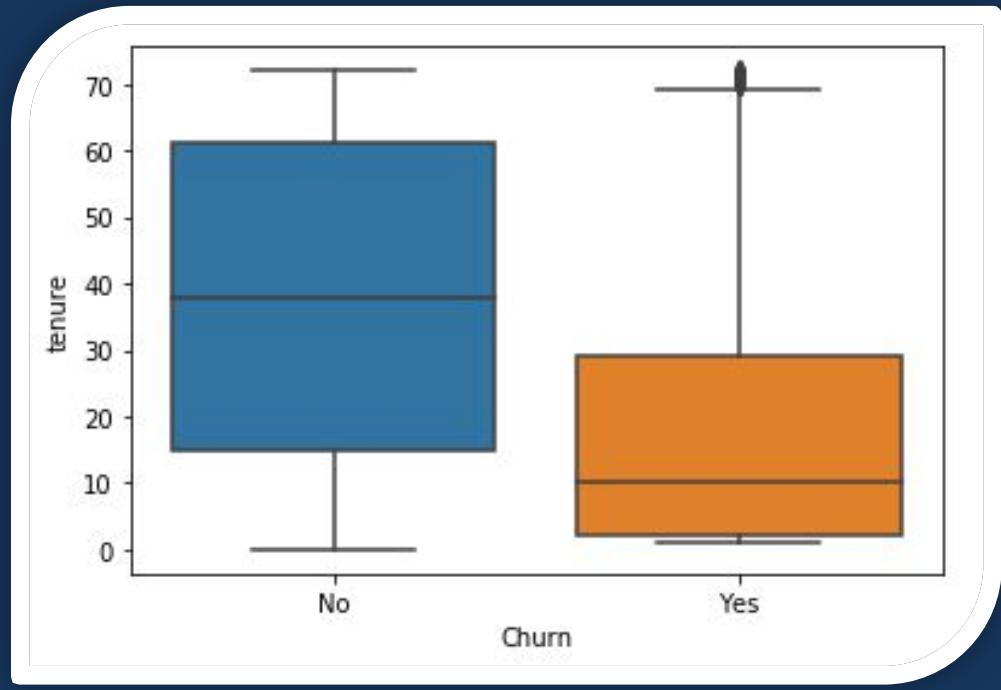


SeaBorn BoxPlot

```
churn = pd.read_csv('churn.csv')
churn.head()
```

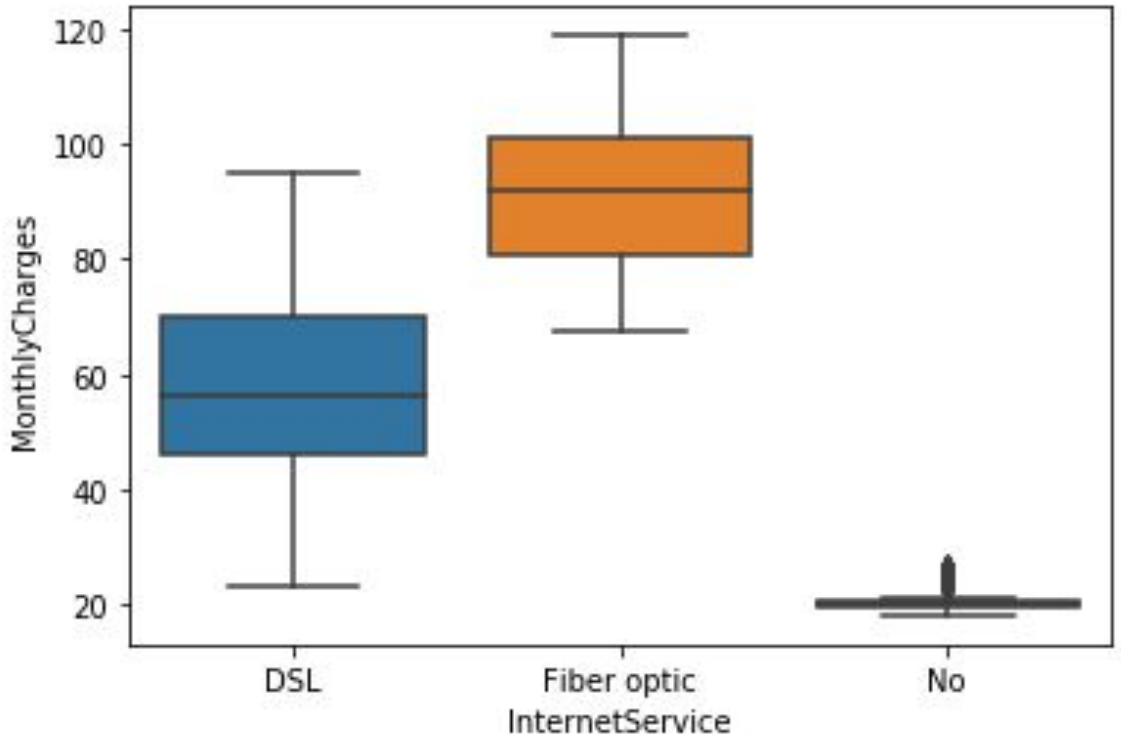
ustomerID	gender	SeniorCitizen	Partner	Dependents	tenure
7590-VHVEG	Female	0	Yes	No	1
5575-GNVDE	Male	0	No	No	34
3668-QPYBK	Male	0	No	No	2
7795-CFOCW	Male	0	No	No	45
9237-HQITU	Female	0	No	No	2

```
sns.boxplot(x='Churn',y='tenure',data=churn)
plt.show()
```



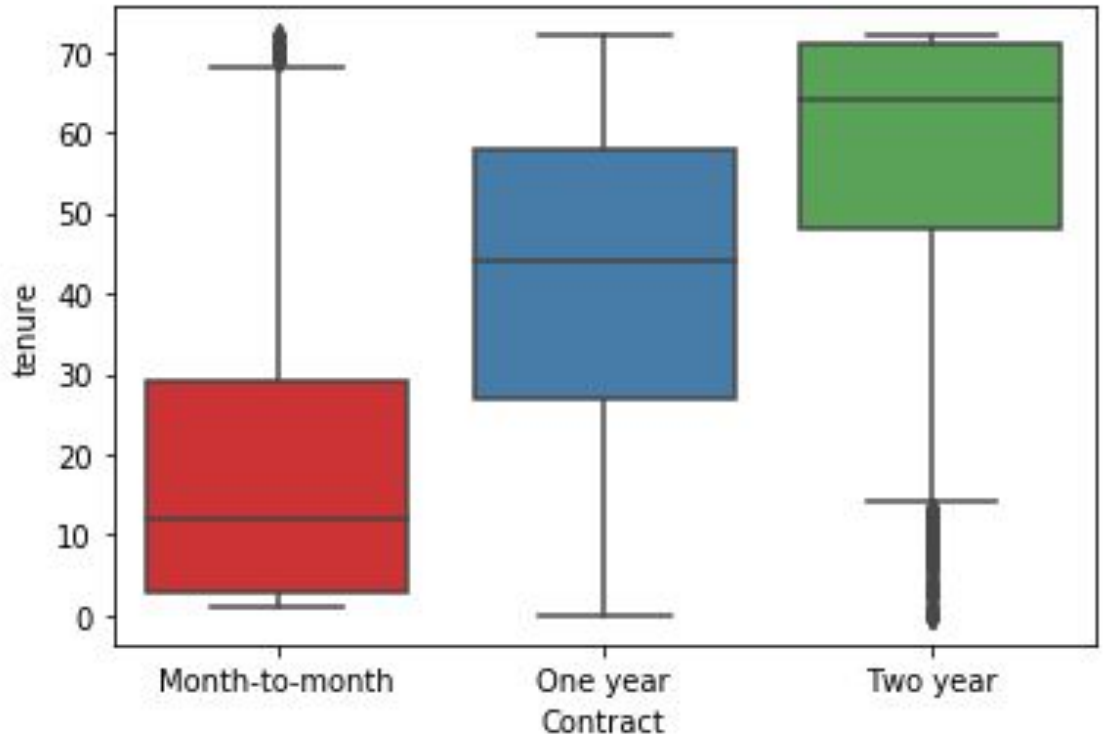
SeaBorn BoxPlot

```
sns.boxplot(x='InternetService',y='MonthlyCharges',data=churn)
plt.show()
```



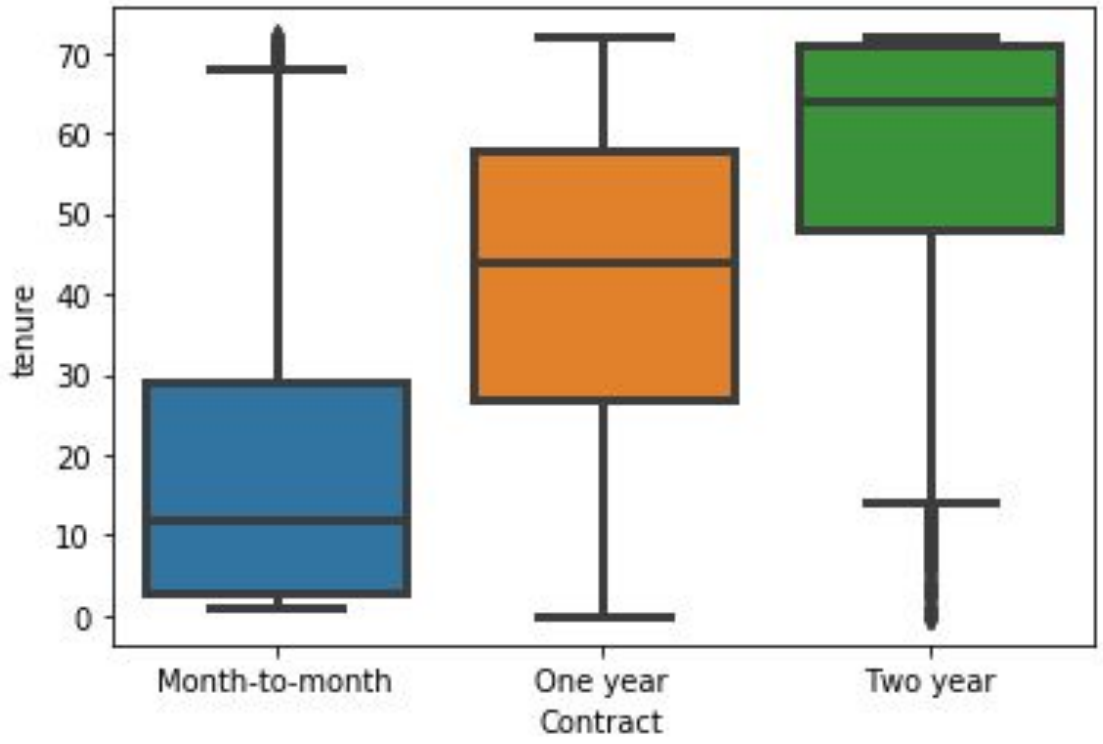
SeaBorn BoxPlot

```
sns.boxplot(x='Contract',y='tenure',data=churn,palette="Set1")
plt.show()
```



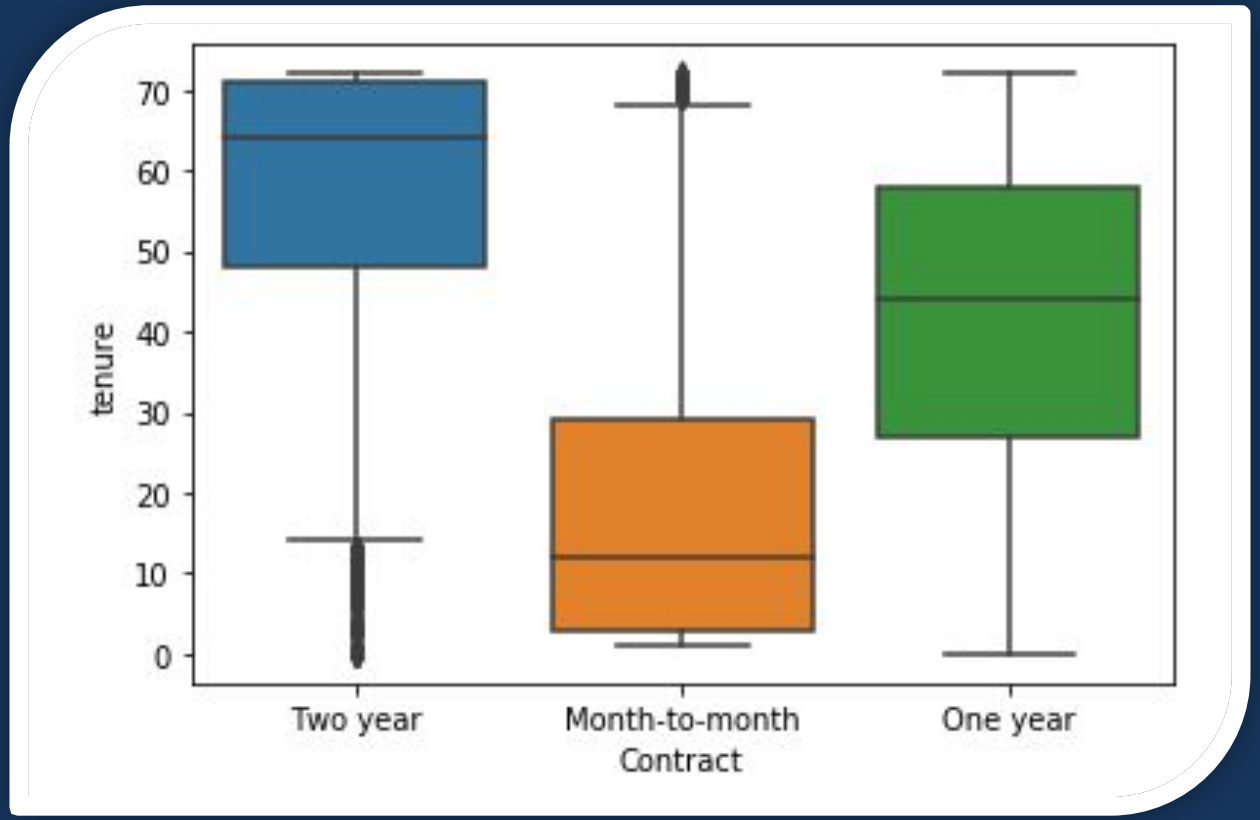
SeaBorn BoxPlot

```
sns.boxplot(x='Contract',y='tenure',data=churn,linewidth=3)
plt.show()
```



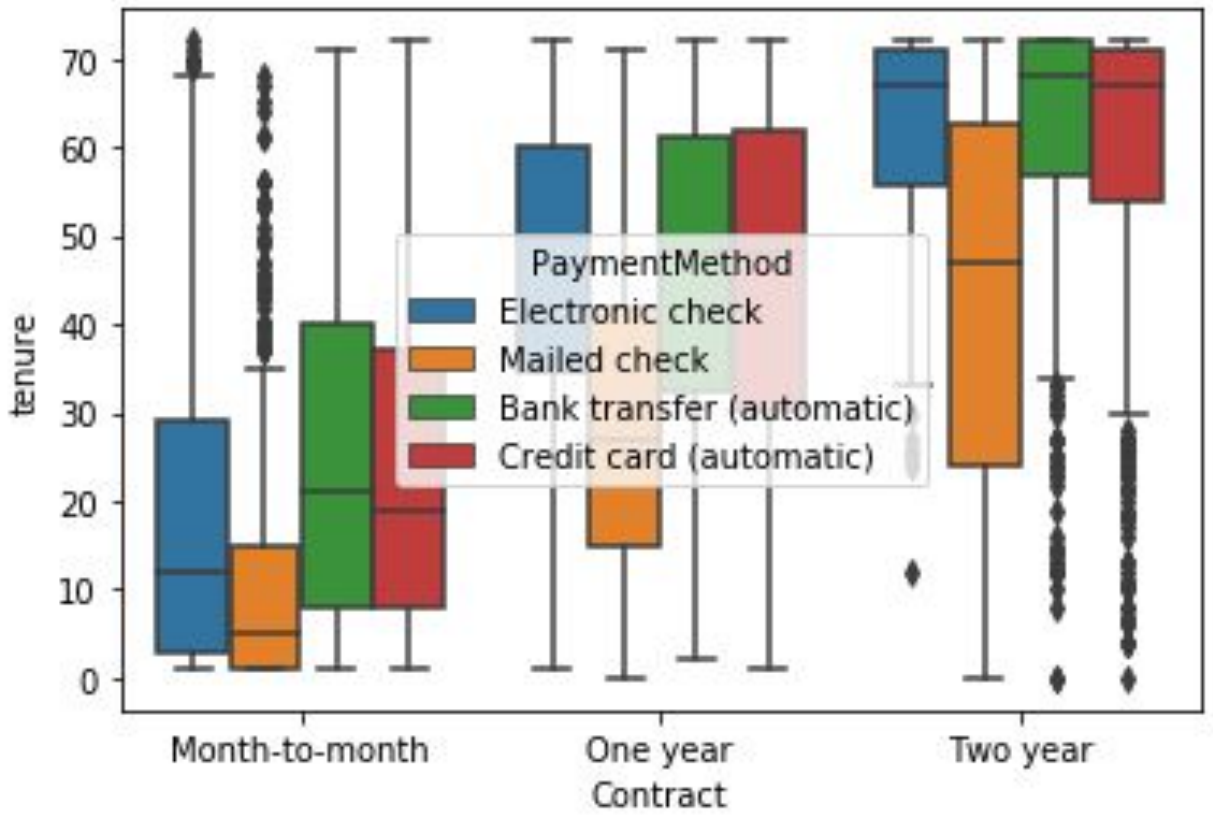
SeaBorn BoxPlot

```
sns.boxplot(x='Contract',y='tenure',data=churn,order=["Two year","Month-to-month","One year"])
plt.show()
```



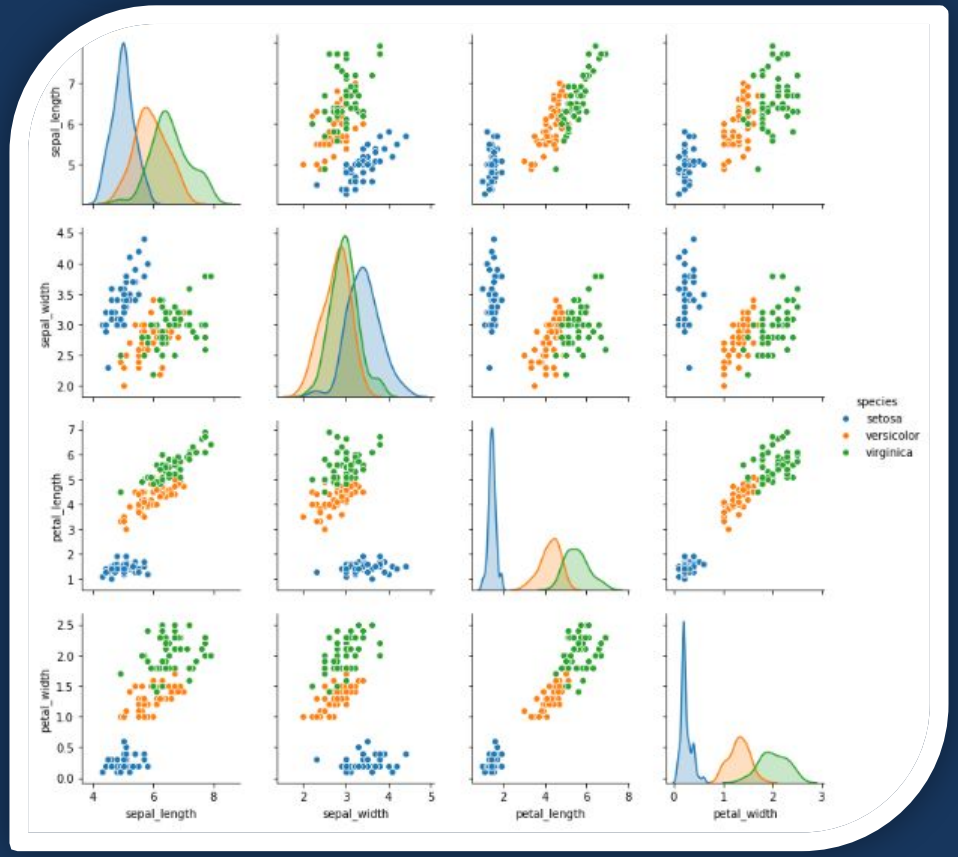
SeaBorn BoxPlot

```
sns.boxplot(x='Contract',y='tenure',data=churn,hue="PaymentMethod")
plt.show()
```



SeaBorn Pair Plot

```
df = sns.load_dataset("iris")
sns.pairplot(df, hue="species")
plt.show()
```



greatlearning
Learning for Life

Case Study

Case Study

We will have a case study on this census dataset

age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex
90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female
82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female
66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female
54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female
41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female

Case Study

```
import pandas as pd
census=pd.read_csv('census.csv')
census.head()
```

age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex
90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female
82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female
66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female
54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female
41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female

```
In [7]: census.shape
Out[7]: (32561, 15)
```

Case Study

```
census['age'].min()
```

```
17
```

```
census['hours.per.week'].mean()
```

```
40.437455852092995
```

```
census['age'].max()
```

```
90
```

```
census['hours.per.week'].max()
```

```
99
```

Case Study

```
census['race'].value_counts()
```

```
White                 27816
Black                  3124
Asian-Pac-Islander     1039
Amer-Indian-Eskimo      311
Other                   271
Name: race, dtype: int64
```

```
census['sex'].value_counts()
```

```
Male      21790
Female    10771
Name: sex, dtype: int64
```

```
census['income'].value_counts()
```

```
<=50K    24720
>50K      7841
Name: income, dtype: int64
```

```
census['workclass'].value_counts()
```

```
Private             22696
Self-emp-not-inc     2541
Local-gov            2093
?                    1836
State-gov            1298
Self-emp-inc         1116
Federal-gov           960
Without-pay            14
Never-worked            7
Name: workclass, dtype: int64
```

Case Study

Renaming Columns

```
census.rename(columns={'workclass':"employment_type"},inplace=True)
```

```
census.rename(columns={'hours.per.week':"hours_worked"},inplace=True)
```

Case Study

Extracting Individual Columns

```
unmarried = census[census['relationship']=='Unmarried']
unmarried.head()
```

```
divorced = census[census['marital.status']=='Divorced']
divorced.head()
```

Case Study

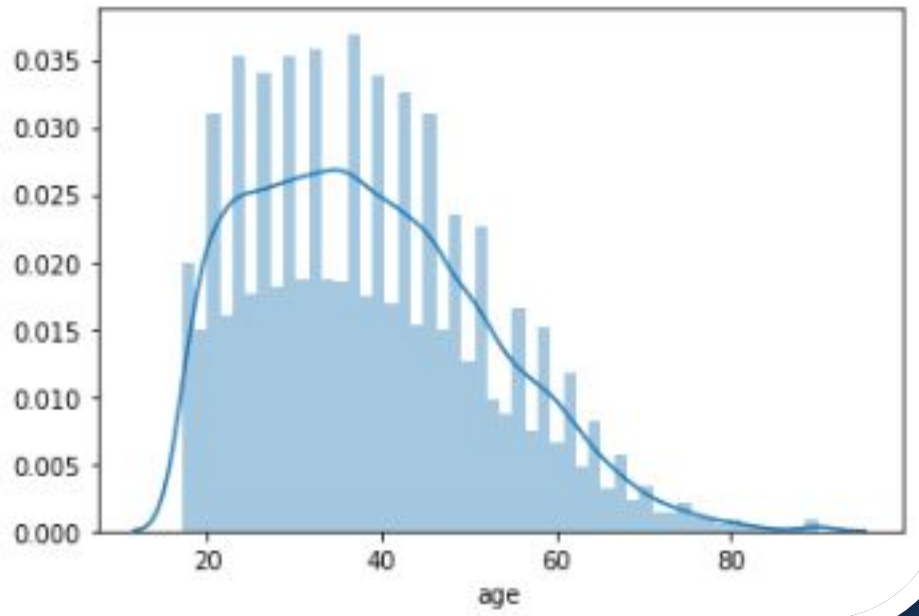
```
old_male = census[(census['age']>50) & (census['sex']=='Male')]
old_male.head()
```

```
white_income = census[(census['race']=='White') & (census['income']=='>50K') ]
white_income.head()
```

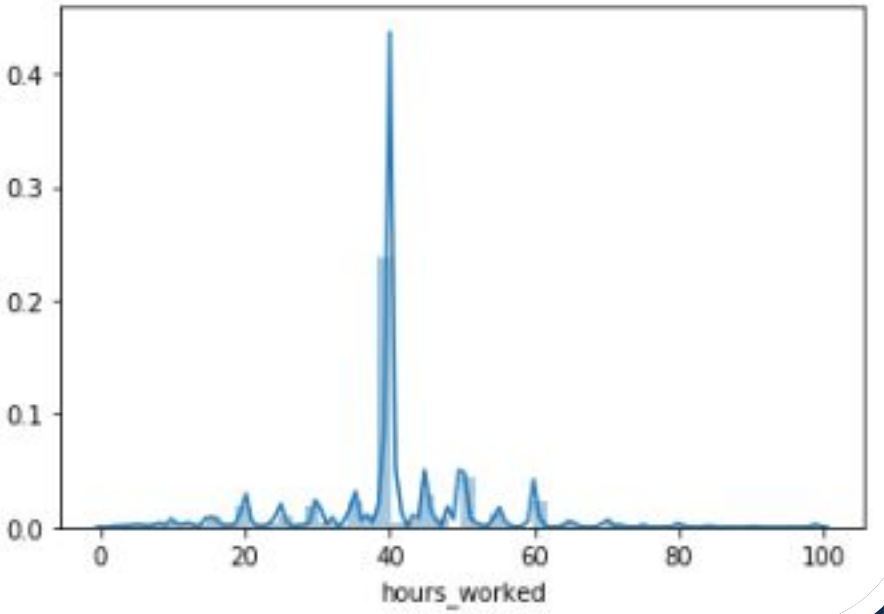
```
master_private= census[(census['education']=='Masters') & (census['employment_type']=='Private') ]
master_private.head()
```

Case Study

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

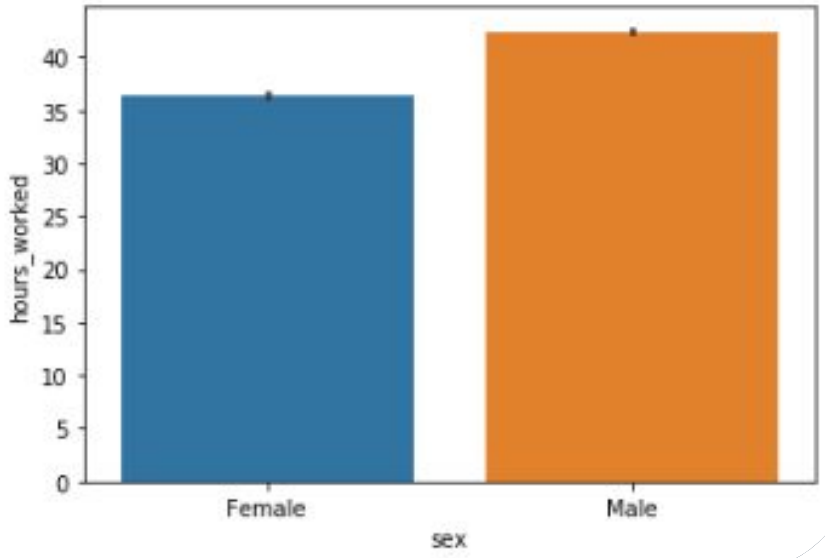



```
sns.distplot(census['hours_worked'])
plt.show()
```

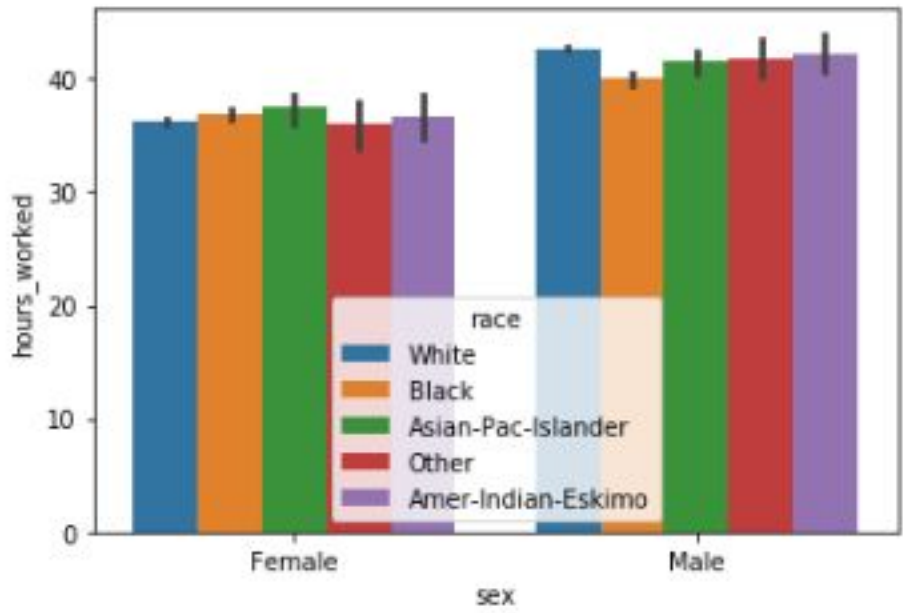


Case Study

```
sns.barplot(x='sex',y='hours_worked',data=census)
plt.show()
```

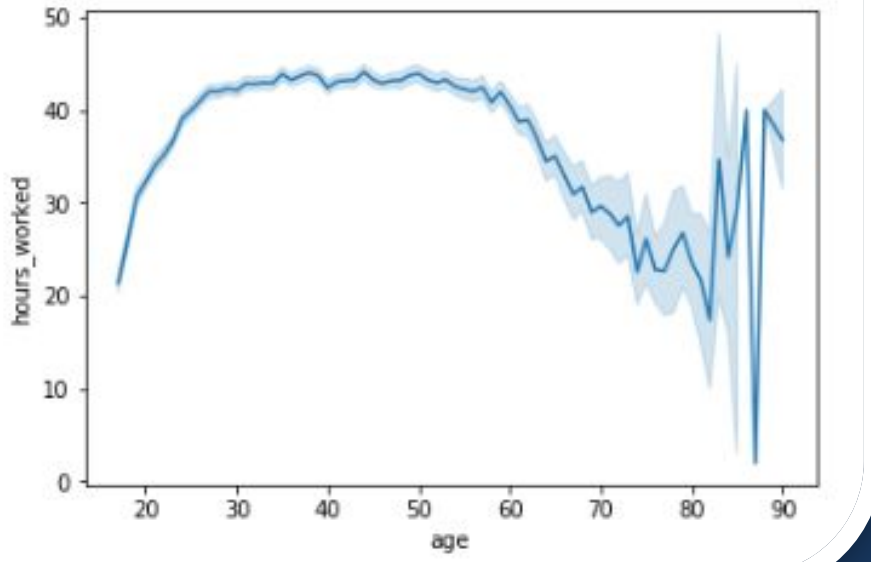



```
sns.barplot(x='sex',y='hours_worked',data=census,hue="race")
plt.show()
```

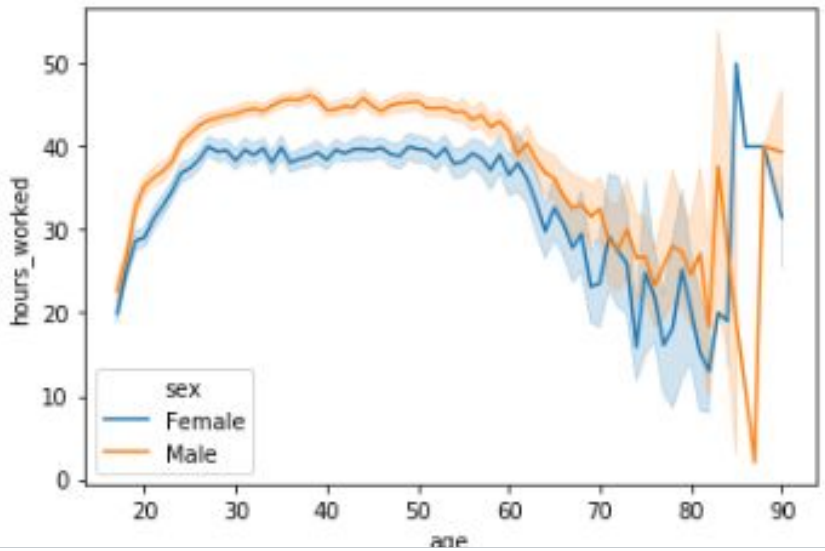


Case Study

```
sns.lineplot(x='age',y='hours_worked',data=census)
plt.show()
```

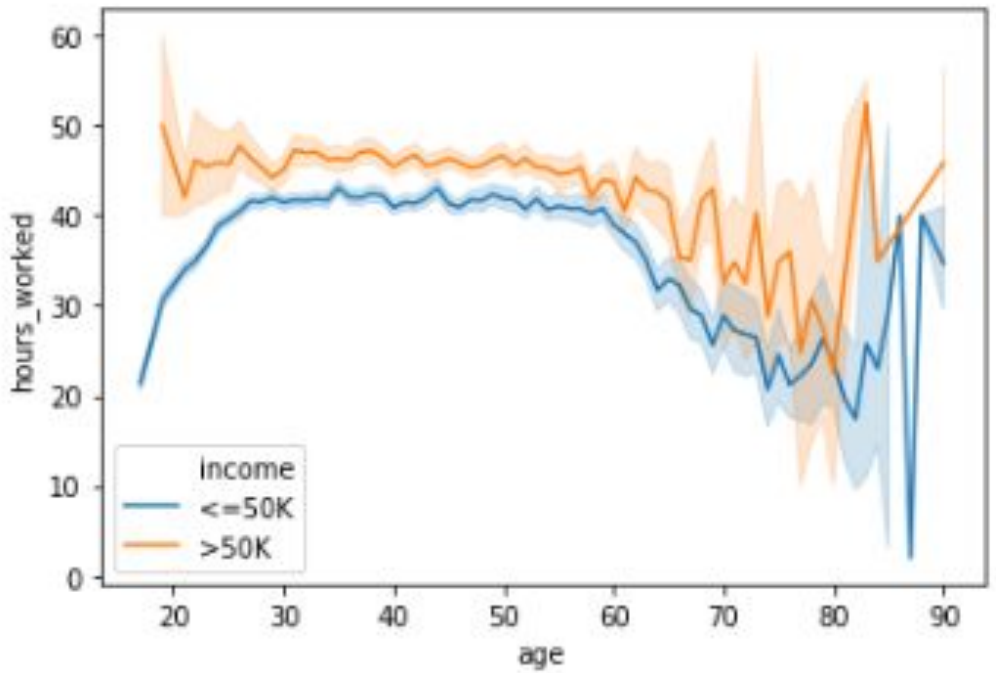



```
sns.lineplot(x='age',y='hours_worked',data=census,hue="sex")
plt.show()
```



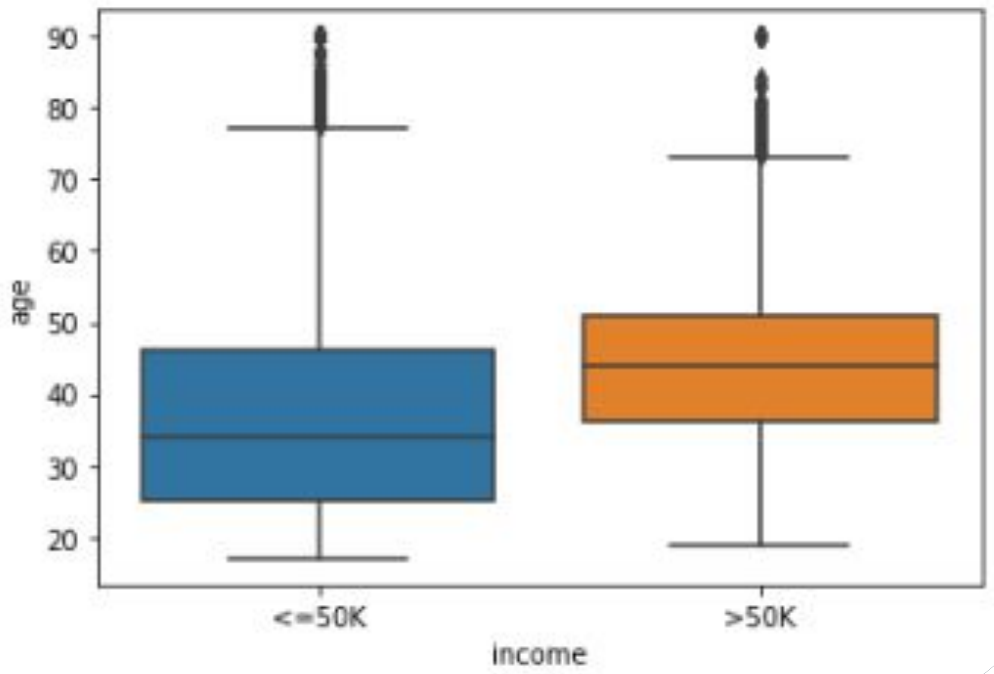
Case Study

```
sns.lineplot(x='age',y='hours_worked',data=census,hue="income")
plt.show()
```



Case Study

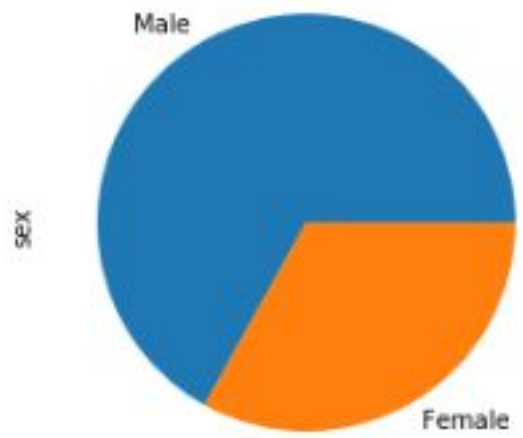
```
sns.boxplot(x='income',y='age',data=census)
plt.show()
```



Case Study

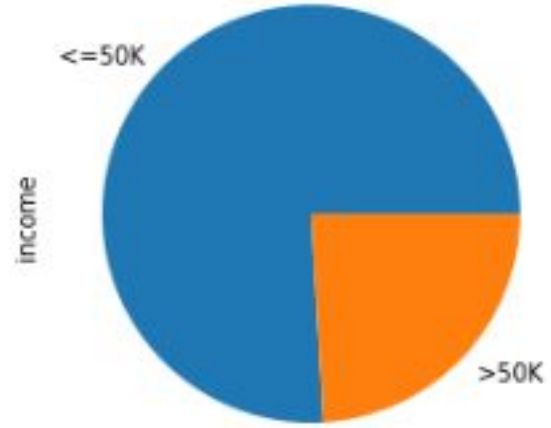
```
census.sex.value_counts().plot(kind='pie')
```

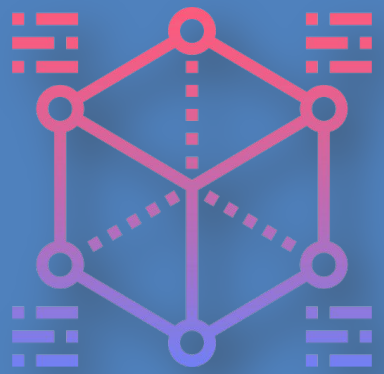
```
<matplotlib.axes._subplots.AxesSubplot at 0x1fe2e5bfcc8>
```

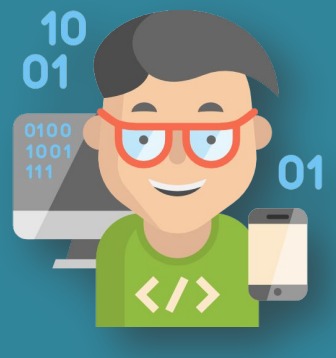



```
census.income.value_counts().plot(kind='pie')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1fe2b6a4188>
```







Thank You