Contents

[Visualization 101 2](#_Toc95447832)

[Need for Visualization 2](#_Toc95447833)

[Basic rules for a good chart 2](#_Toc95447834)

[Object Oriented Programming Terminologies (Class/Objects/Methods) 8](#_Toc95447835)

[Matplotlib - I 18](#_Toc95447836)

[3 Layers of a chart in Python Matplotlib 18](#_Toc95447837)

[MCQs – 20](#_Toc95447838)

# Visualization 101

## Need for Visualization

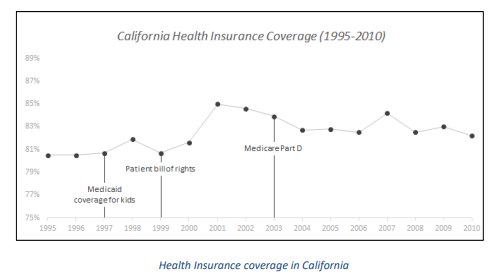
* To be able to present a story from given data to various stakeholders interested in insights
* These stakeholders can be –
  + A data scientist looking for patterns in data to help him build a classification or regression model
  + A business person (Manager/Executive/ CXO etc.) looking for some actionable insights from the data
  + To look for deep meaningful patterns which can be used in presentations/reports etc. for a broader population

In this week, we will learn to process raw data using NumPy/Pandas to our suitability, and present it into various insight rich meaningful charts using matplotlib/Seaborn.

## Basic rules for a good chart

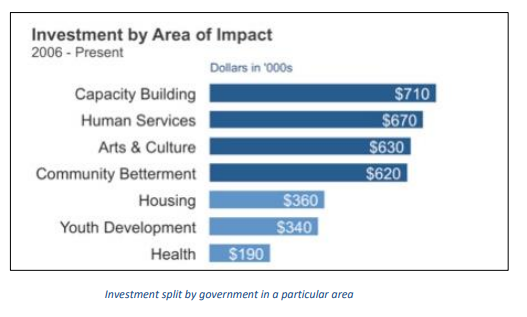
* Always chose the appropriate chart as per the data to be presented (3 min) -
  + For example, timelines are best represented using line charts

(Year on year sales of a company, Monthly marketing expenditure etc.)

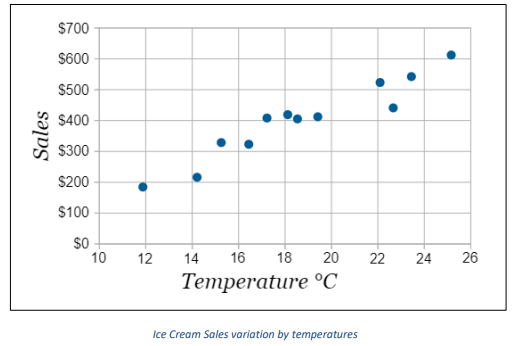


* + Category-wise aggregations are best represented using bar charts

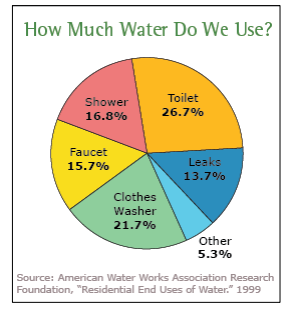
(Annual sale of different products, Region-wise customer counts, Marketing expenditure on different channels etc.)



* + A pattern between 2 quantitative variables can be represented with a Scatter Plot



* A percentage split across categories can be represented by a pie chart

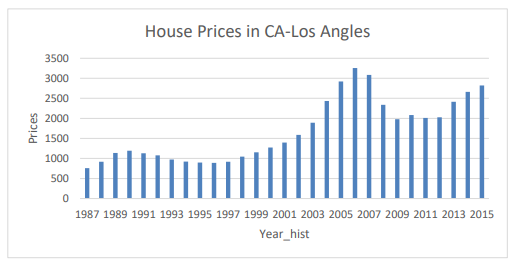


* These are most commonly used charts because these are easy to understand and provide great insights. But it’s always good to explore new charts on our own and enhance our armour. Few charts which must be explored are - Stacker Bar chart, Donut Chart, Waterfall Chart, Pareto chart, Tree map.

We should choose a chart which is easy to infer, fits the story best and visually appealing so that it attracts focus

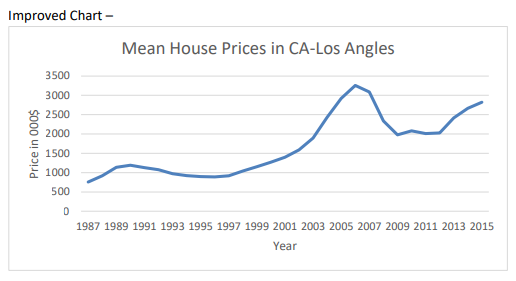
* We must always specify the axes titles clearly
  + Never use coding terminology such as “count\_of\_customers” in axes names. It should be like “Count of Customers” instead
  + Never use incomplete axes title, it should be exhaustive enough so that anyone having little or no context about the data should be able to understand the point that we are trying to make.

For example, “Price” on Y-Axis in below chart doesn’t tell what the units and currency of price are.



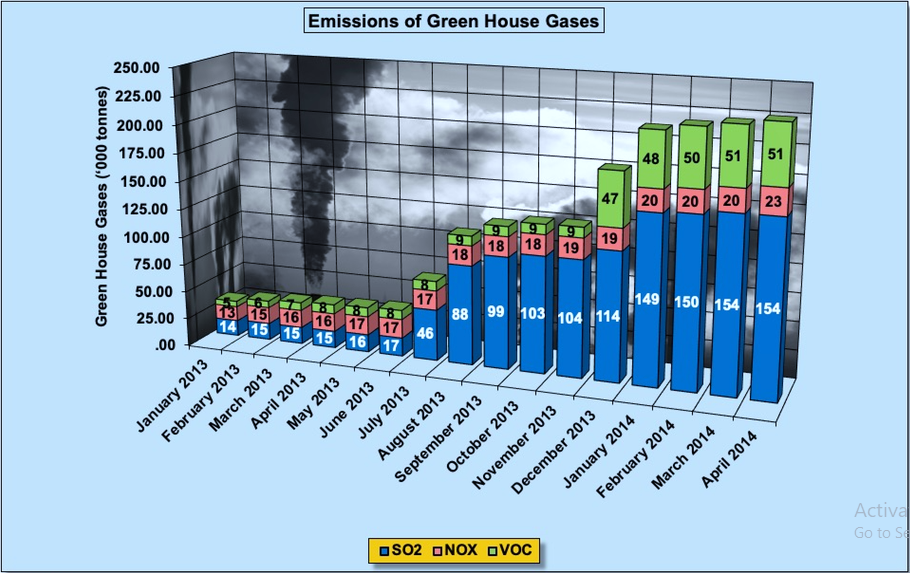
Issues in above chart –

* + - Axis titles can be improved.
    - The X-Axis uses coding terminology, and Y-axis misses units and currency details
    - A line chart would show trend in a better way
    - There is no information about what is the size of the house we’re talking about. It can be mean house prices, median house prices, house price of a particular standard sized home etc.
    - The chart is open to interpretation, it’s not exhaustive enough.



* Chose colors wisely -
  + Red usually represents something –ve/ shortage/ decrease
  + Green usually represents something +ve/ growth
  + Refrain from using Green/Red in neutral scenarios
  + Do no chose flashy colors for no reason ( **Turqoise**, Highlight Yellow etc.)
  + Do no chose extremely light colors which are hard to see or distinguish amongst.
  + Refrain from using the theme color of a competitive company.
* Be Consistent–
  + If you’re using multiple visualizations in your report/notebook/presentation, make sure the fonts, terminologies and color themes are exactly same everywhere.
* The visualization should not be ambiguous. It should have a clear objective message.
* Know your audience -
  + Empathize with your audience and build a chart accordingly. The same chart can be most effective against a particular audience and really poor against another
  + A data scientist, subject matter expert etc. can understand complex densely informative visualization. These people have good understanding of the technicalities and context, and thus look for deeper insights. Examples – Box Plot, Histogram, Heat Maps etc.
  + In contrast, a high level business stakeholder or a general audience might not understand a complex chart having too much information. Such audience usually have limited time as well, and it is best to keep visualizations simple and precise. Basic line chart, pie chart, bar chart can be wise options for such audience.
  + There will always be a trade-off among many factors while deciding on a vision for the chart. A small research on “The Visualization Wheel” might be useful to understand different trade off components to think about in such scenarios. (For now, let’s keep the discussion on this topic, but students should explore about The Visualization Wheel on their own)
* Maximize Data to Ink Ratio
  + It’s a concept by Edward Tufte explained in his book *The Visual Display of Quantitative Information*
  + The principle behind is simple – Above all else show data!
  + Data Ink Ratio = Data Ink / Total Ink used
    - Data-Ink: Non-redundant ink which is used in a must have component of the chart. It builds the core of the visualization and cannot be removed
    - Non-Data Ink: Additional ink used in the chart for superficial or artistic purpose, and doesn’t add any value to the insights
    - Total Ink: Data Ink + Non data ink

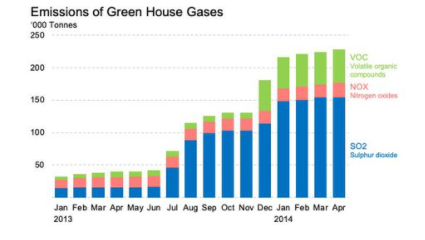
A bad chart with low data ink ratio -



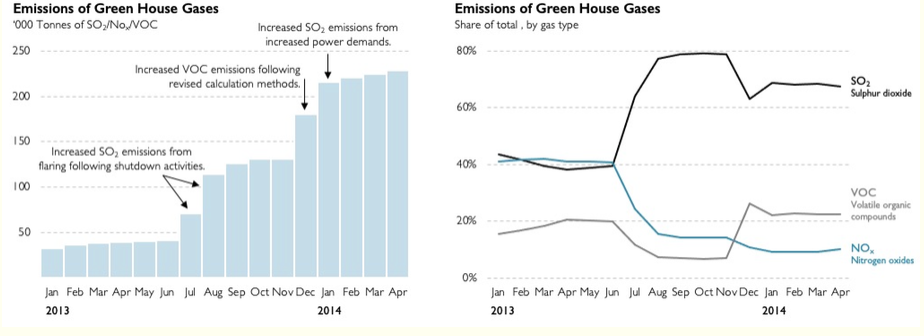
Possible alterations –

* Remove 3D effect
* Remove background Image
* Remove background fills
* Remove all shape effects, outlines
* Remove bold effect
* Remove redundant labels
* Remove unnecessary gridlines
* Resize graph to a compact shape

A much better chart with better data-ink ratio :



A more fine tuned visulisation can be as following with better data-in ratio :



Optional Read –Visualization wheel : <https://ryanwingate.com/visualization/guidelines/visualization-wheel/>

Optional Read – Data-Ink Ratio : <https://simplexct.com/data-ink-ratio>

# Object Oriented Programming Terminologies (Class/Objects/Methods)

OOPs is not a core necessity for learning Matplotlib because there are much simpler functions in Matplotlib to create visualisations.

But we are touching upon it because OOPs is the underlying architecture, and knowing OOPs will help a lot in understanding various errors and their resolution on various forums like Stackoverflow.

For this purpose, we will touch-base upon some OOPs terminologies in this module, the students are advised to explore in depth on their own.

OOPs is an approach to problem solving which is based on 4 main pillars –

o Abstraction

o Polymorphism

o Encapsulation

o Inheritance

We’ll discuss about these 4 again in next 10 minutes, before that we need to get an idea about what a Class, Object and Method is.

In OOPs, each problem is dealt with the use of Objects. We can create multiple different objects of a particular class. Classes are the blueprints for these objects. We can define Class in ways which will govern the way Objects can interact among themselves.

OOPs allows for defining Classes and their Objects. It also enables Class Inheritance which is a key concept for layered architecture which Matlplotlib uses.

(These terms might be too much for someone having no exposure to programming or OOPs. But that would not be an issue, as in this course we would need limited exposure to OOPs. OOPs is a vast subject on its own, but here we will discuss only the most fundamental OOPs terms in order to make out Matplotlib journey easier and more exciting.)

**Class –**

A class is a user-defined data structure. It is a template or a **blueprint** for an object. It mainly consists of 2 parts – *Attributes* and *Methods*.

A class can have multiple objects, and all the objects will obey the blueprint defined in class definition.

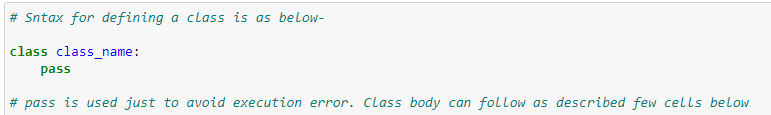
Attributes – These are the properties a class object will have. For example

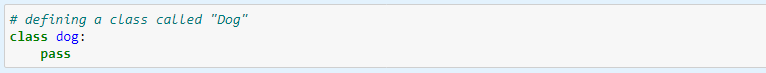
A **Person** class can have Name, Age, Address, Qualifications, Skillsets as Attributes and Walk, Travel, Work, Sleep, Date a girl as Methods.

A **Store** class can have Store ID, location, Revenue, Foundation year, Customer base etc. as Attributes and Open, Close, Promotion as Methods.

A **Dog** class can have Name, Weight, Birth year as Attributes and Eat, Sleep, Bark as Methods.

*Python syntax for defining a Class:*

**

**

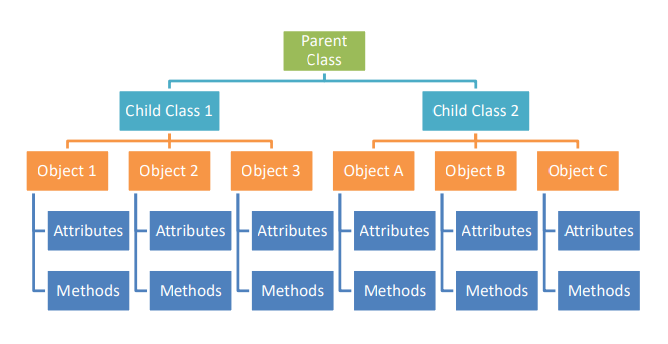
**Objects –**

Objects are instances of a class. While the class is just a blueprint, an object is the actual entity which exists and interacts with other objects. For example, the objects of Dog class would look like –

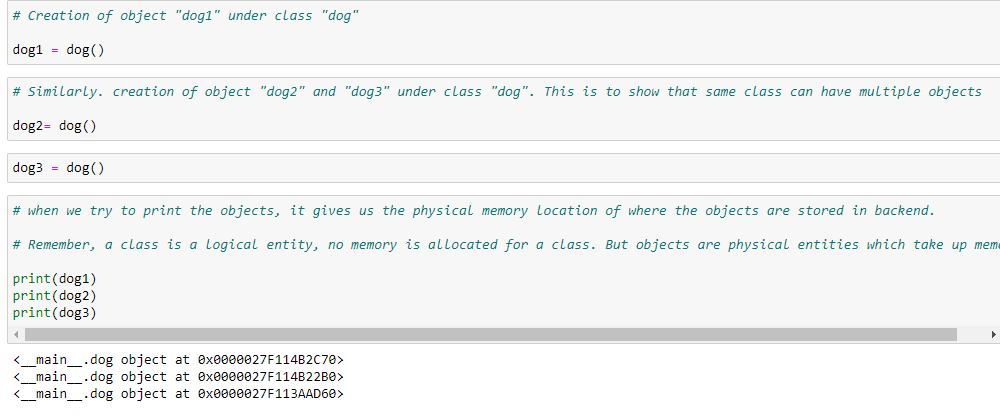
{Shubham, Age: 27, Address: Chandigarh, Skills: Data Analyst}

{Akshat, Age: 41, Address: Delhi, Skills: Data Scientist}

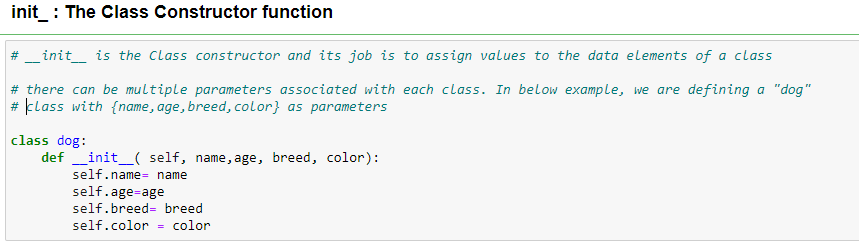
{Shweta, Age: 56, Address: Bangalore, Skills: Housewife}

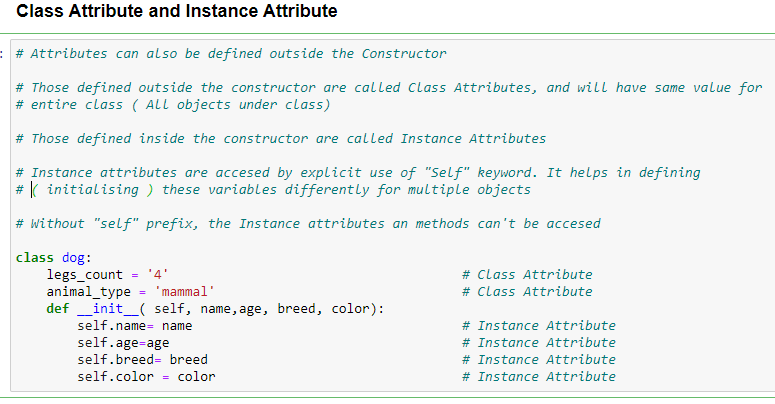


*Python syntax for Objects:*



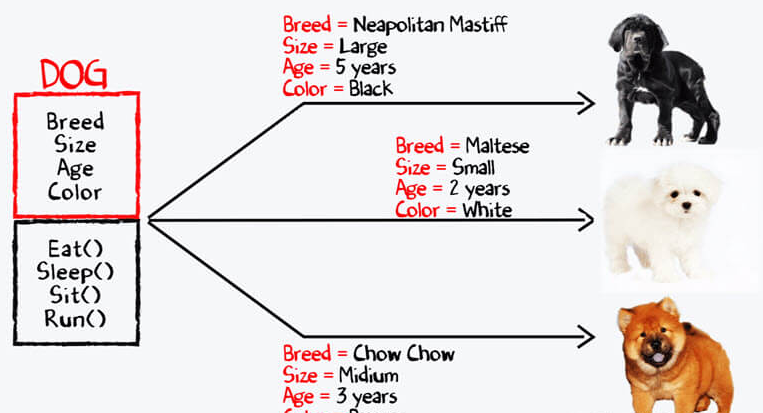
Class constructor (\_\_init\_\_ function):



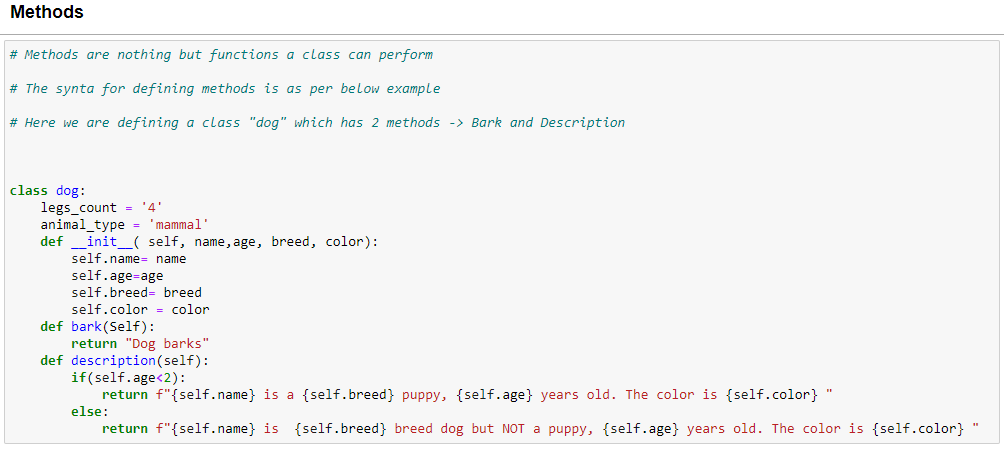


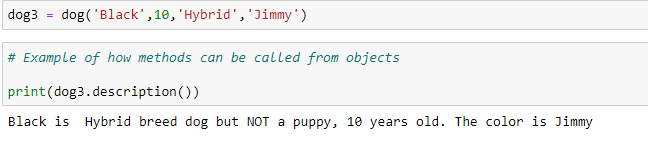
**Methods –**

Functions associated with a class are called its methods. Only objects of the class will have access to these functions (methods).



*Python syntax for Methods:*





**4 Pillars of OOPs:**

1. **Inheritance:**

A class can inherit Attributes and Methods of an another class known as Parent class. In other words, there can be a hierarchy of classes where there can be multiple child classes of a parent class and those child class can also have their own child classes.

Let’s take an example of transportation vehicle as below –

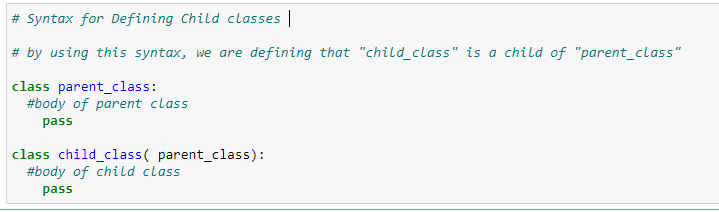
Vehicles: Parent Class

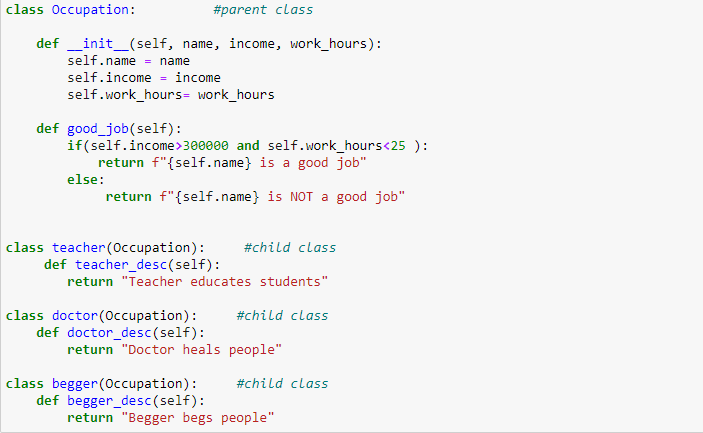
Commercial, Non-Commercial: Child classes of Parent class, Parent classes for all under those

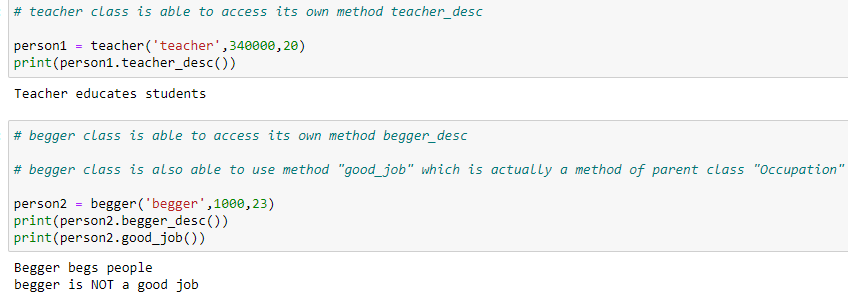
Similarly, Truck, Bus, Cab etc. are Parent/Child classes as well.

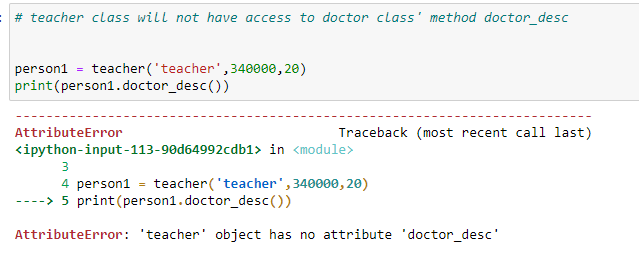
Only the highlighted blocks are Objects, and rest are Parent/Child Classes

Python example showing Inheritance:





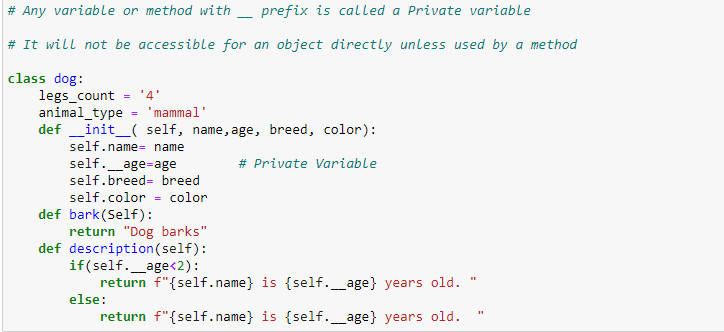


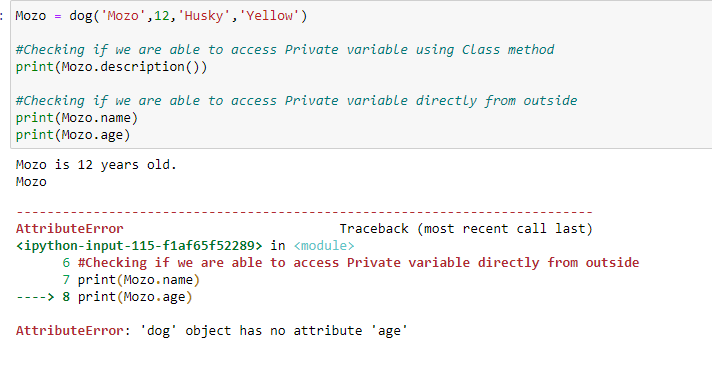


1. **Encapsulation:**

It is a technique used to ensure security. OOP allows for defining private attributes/method in a class which are not easily accessible.

In Python this is done by placing “\_\_” before an attribute name or method while defining.



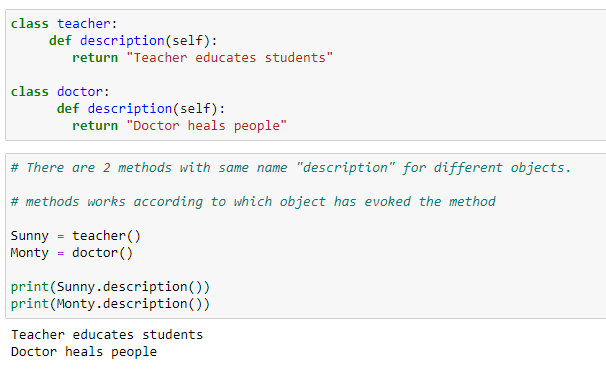


1. **Polymorphism:**

Poly means many and Morph means forms.

Polymorphism by name indicates that a single entity can take multiple forms in OOP.

For example, there can be methods with the same name in 2 different classes, but both methods will act differently as according to which object has invoked the method.



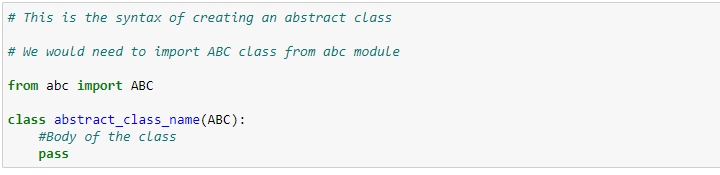
1. **Abstraction:**

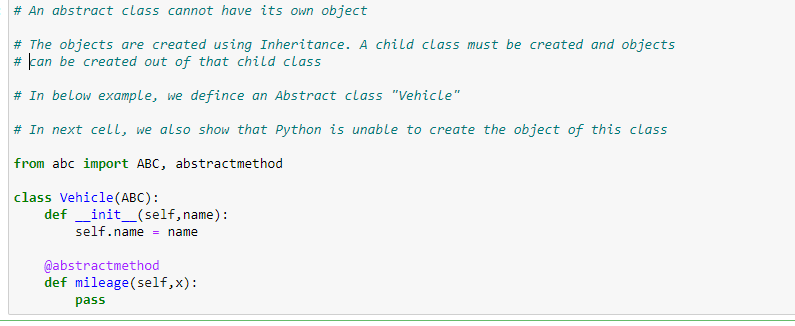
Using abstraction, the functionality of a solution is brought to the forefront while hiding away the inner details.

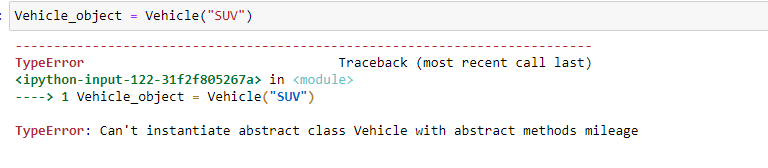
It is similar to riding a bike. We ride a bike by just adjusting clutch, brake and acceleration, without the deeper know-how of how actually the engine is generating power.

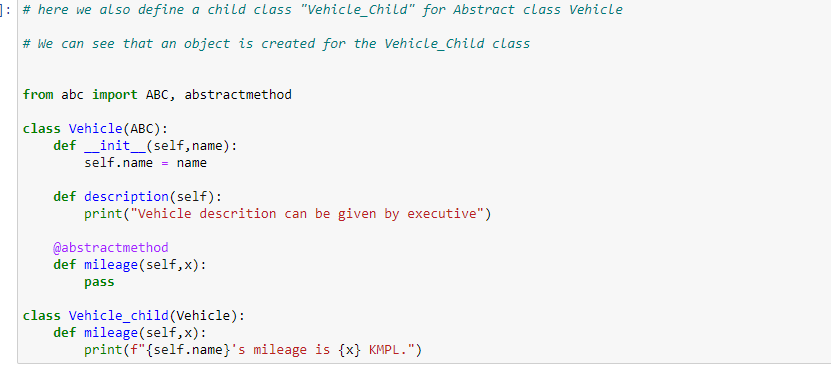
In Matplotlib we’ll learn many functions and their use cases, but it is not always important to know what goes behind on the backend.

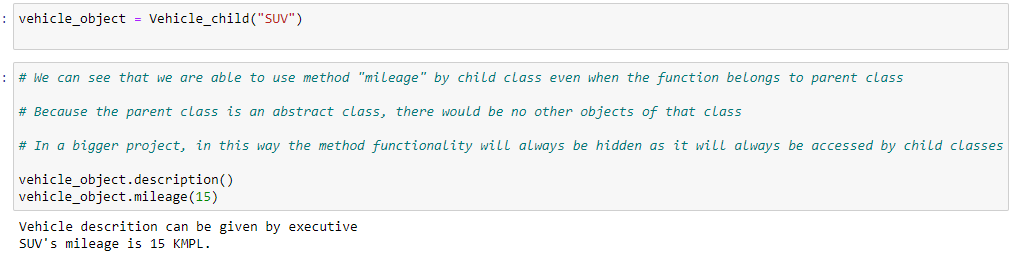
We will witness all of these in action in our journey with Matplotlib, and hence it is sufficient for now to have this level of OOP understanding for now.











# Matplotlib - I

## 3 Layers of a chart in Python Matplotlib

1. **Backend:**

A backend, as the name suggests, is the background bridge between Python and Matplotlib. It ensures that the commands/codes written in Python are rendered accurately in the background and a visual is created.

In short it is the “black box” where all the dirty computations happen in the background so that a developer can focus on the high level details of the chart instead of complex programming.

1. **Artist Layer:**

A chart is nothing but an arrangement of some drawing components on a canvas. Artist layer manages and allows us to edit or workaround these components to build the desired visual.

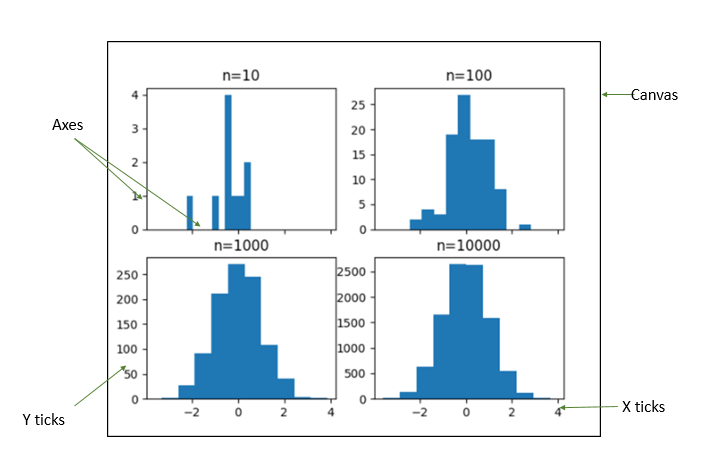
These components are –

**Axes** - Each plot has an X-Axis and a Y-axis. There is an axis object in Matplotlib which enable us to change the range of axes, titles, ticks etc. We would be dealing a lot with this Axes object.

**Subplots**- Each visual can contain one or more subplots. A subplot is a smaller chart within the main canvas. By default, a canvas has only 1 subplot which is called (1,1,1). Often, for better storytelling and comparison purposes, we try to show multiple charts alongside.

**Shapes** –Lines, Rectangles, circles etc.

**Data points / Collections –** Actual data which is fed to the Matplotlib to give us a graphical representation.



The picture demonstrates 4 subplots drawn on a canvas. Some elements like axes, xticks, yticks are marked for easier understanding.

So now we know that there is a Backend layer, which is a black box and does the actual task of converting our codes into a visual.

And there is an Artist layer on top of backend which helps us manipulate with every little component of a visual like axes, titles, tick marks etc. Lets’ talk about the third layer now –

1. **Scripting Layer-**

It’s a high level layer built on top of artist layer to simplify our interaction with the Matplotlib further.

It works like a magic and does a lot of complex tasks in simple functions for us. Of course, its simplicity comes with some limitations, and we would require to use the Artist layer objects often.

The scripting layer that we would use in this course is *pyplot*.

**Summary:**

There is a backend, which renders the code to create an image.

There are Artist objects on top of backend which define the how data is arranged and shown to the user.

And Scripting layer on the top which interacts with the Artist layer in simpler ways.

# MCQs –

1. **Which one is the correct statement?**

A. Classes are real entities which take space and Objects are logical entities.  
B. Objects are real entities which take up memory and Classes are logical only entities.

C. Class and Object both are real entities  
D. Both are logical entities

1. **A method is a/an -**

A. attribute within class  
B. object within class  
C. argument within class  
D. function within class

**3. Which is true about private variable of a class?**

A. It can never be accessed

B. It can be accessed as any variable, but only by the objects of the class

C. It can only be accessed by using method of the class objects

D. None

**4. A class is an instance of the object.**

1. True
2. False

**5. A child class can access methods of its parent class.**

1. True
2. False

**6. A parent class can access methods of its child class.**

1. True
2. False

**7. There are 3 classes - Products, Clothing, Appliances**

**All these 3 classes can use the *Show\_advertisement* method of some other class Appliances.**

**This is an example of -**

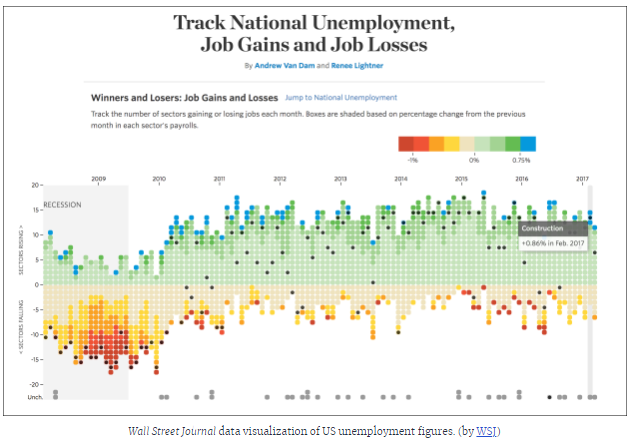
1. Abstraction
2. Polymorphism
3. Encapsulation
4. Inheritance

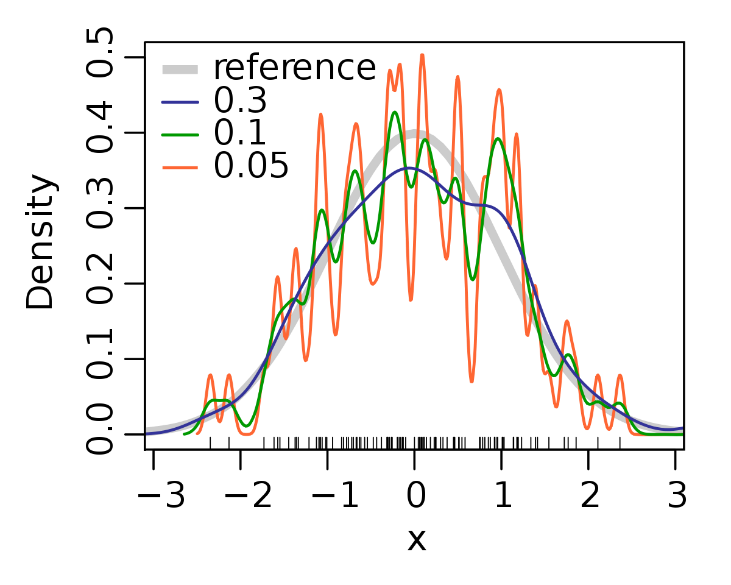
**8. There are 3 classes - Products, Clothing, Appliances**

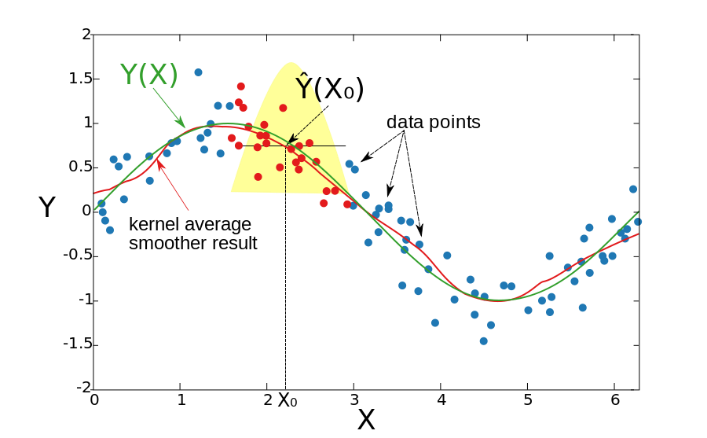
**All these 3 classes can have their own *Billing* method which is different from each other. This is an example of -**

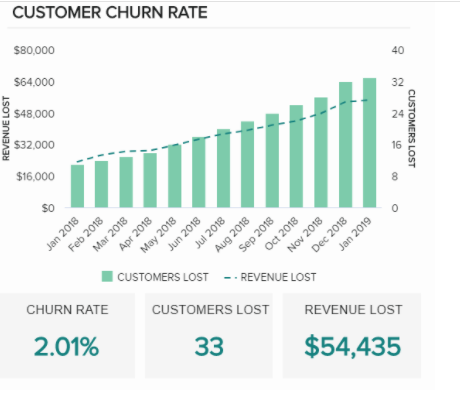
1. Abstraction
2. Polymorphism
3. Encapsulation
4. Inheritance

**9. Which of the following charts is most appropriate for a high level business stakeholder -**

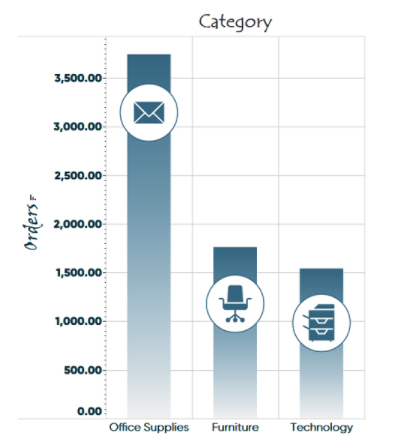
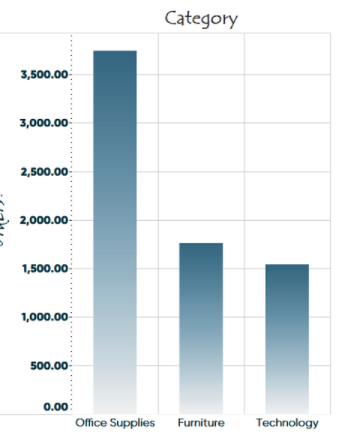
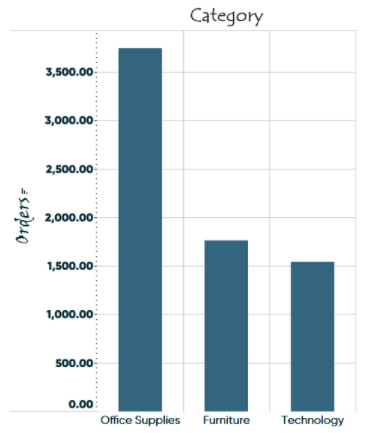
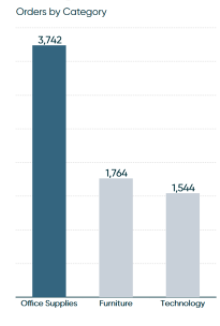
1. 

B. 

C. 

D. 

**10. Which of the following have maximum data-ink ratio-**

1. 
2. 
3. 
4. 

Tasks –

* MCQ completion
* Pre read
* Lesson Plan
* Assignment Solution
* Homework
* Adding snippets
* Review doc