PYTHON
MACHINE LEARNING
FOR BEGINNERS

THE BEGINNER'S GUIDE TO BIG DATA ANALYTICS, DATA SCIENCE, DATA ANALYSIS WITH MACHINE LEARNING AND PROGRAMMING CODE WITH PYTHON

LEONARD DEEP

Python Machine Learning for Beginners

The Beginner's Guide to Big Data Analytics, Data Science, Data Analysis with Machine Learning and Programming Code with Python

LEONARD DEEP

Download the Audio Book Version of This Book for FREE

If you love listening to audio books on-the-go, I have great news for you. You can download the audio book version of this book for **FREE** just by signing up for a **FREE** 30-day audible trial! See below for more details!

Audible Trial Benefits

As an audible customer, you will receive the below benefits with your 30-day free trial:

- FREE audible book copy of this book
- After the trial, you will get 1 credit each month to use on any audiobook
- Your credits automatically roll over to the next month if you don't use them
- Choose from Audible's 200,000 + titles
- Listen anywhere with the Audible app across multiple devices
- Make easy, no-hassle exchanges of any audiobook you don't love
- Keep your audiobooks forever, even if you cancel your membership
- And much more

Click the links below to get started!

For Audible US

For Audible UK

For Audible FR

For Audible DE

Table of Contents

Introduction

Welcome and much appreciation for downloading this book. The book will cover the entire Python programming from an absolute beginner perspective and place more emphasis on the first three chapters to enable the reader to get furnished. Chapter 3 of the book covers how to install and run Python IDE and including your first “Hello World Program”.

This book is written with the understanding that the student is completely new to Python programming. For this reason, the author takes measures not to assume or leave out any part. Unlike other books, the author decided to split content meant for the first chapter into three chapters to gradually introduce the learner into Python programming. The author explains systematically how to get started, keywords, statements, variables, data types and type conversion among other basic areas of Python programming. After providing examples, the author also provides exercises. The exercises are carefully selected and will not overwhelm an absolute beginner. They are meant to help the learner systematically build skills and confidence in Python programming. Once through with the book, the learner will be more than ready to handle any Python programming challenge. Let us begin.

Chapter 1: Introduction to Machine Learning

The topics discussed in this chapter come from an analysis of the main requirements in data scientist jobs from popular technology companies. The topics are as listed below:

- Linear Regression(Simple)
- Linear Regression(Multiple)
- Evaluating Performance of Models
- Hierarchical Clustering
- KNN
- Kernel SVM
- Clustering K-Means
- Principle Component Analysis-PCA
- Pandas (Python Library for Handling Information or Data)

Artificial intelligence, ML, and data science are some of the trending subjects in the technology field today. Bayesian analysis and data mining are the ones currently trending and this has contributed to the increasing demand for ML (ML).

ML is a field that focuses on the programming of computer systems hence causing them to learn and improve with expertise automatically.

ML Algorithms Applications

- Data mining
- Games
- Robotics
- Expert computer systems
- Recognition of patterns
- Vision processing
- Language processing
- Stock market trend and weather forecast

Procedures involved in ML

An efficient ML project entails the following steps:

- Definition of the problem
- Preparation of data
- Evaluation of algorithms
- Improving the results
- Presentation of the results

The perfect way of starting on ML using Python is working through a project from the beginning to the end and cover the major steps such as summarizing data, evaluating algorithms, loading information, and predicting. This will provide you with a replicable technique that can be used for each dataset. Adding further data to improve the results is also an available option.

To understand machine learning you are required to understand how files work in Python. The following section discusses Python files.

Python Files

Files are used for future storage. To read from a file we need to open it and once through with it, we have to close it to free the resources.

File Opening

The inbuilt function open() in Python is used to launch a file.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
file_name=open(“lesson.txt”)
file_name=open(“D:/Tutorials/lesson.txt”)
```

Exercise

a. Locate a text file on your computer and open using the Python open()
b. Locate a doc/docx file on your computer and open using Python open()
c. Locate an excel file on your computer and open using Python open()

Additionally, Appending ‘a’, write ‘w’, or read ‘r’ the file helps indicate the mode when launching the file. The modes are determined by the context and can be changed as the need arises. Python also allows us to indicate binary or text mode depending on our programming needs. By default, the file is opened in text read mode. Images and non-text files are handled in the binary mode.

Python File Modes

The ‘r’ is used to launch the file for reading and is the default. The ‘w’ is preferred when launching writing permissions. For exclusive file creation, we use the ‘x’ mode. The operation will be unsuccessful if the file already we want already exists. The other mode, ‘a’ is used to launch a file for adding data at the file end while retaining earlier content. For the ‘t’ mode, it will launch the file in default text mode. The ‘b’ mode is used to launch the file in binary mode. Lastly, the ‘+’ mode is used for launching the file to allow reading and writing.

Exercise
a. Open an existing .txt file on your computer in Python’s read mode.
b. Open the same text file in a. in Python’s write mode.
c. Open an existing image on your computer in binary mode.

Specifying the File Encoding Type

It is encouraged to specify the file encoding type as it varies from operating system platform to another and may produce different results if not specified.

Example
Start IDLE.

Navigate to the File menu and click New Window.
Type the following:

```
file_name = open("my text.txt",mode = 'r',encoding = 'utf-8')
```

Closing a File in Python

The method close() is used.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
file_name=open(“my text.text”, encoding=’utf-8’)
file_name.close()
```

Exercise

a. Locate a .txt file on your computer, open it using open() and close it using close()

b. Locate a .doc file on your computer, open it using open() and close it using close().

Important

This approach will exit without closing a file should an exception occur when performing some operation with the file. We will tackle what an exception is but consider it as an unexpected error during execution of a program. The try..finally block is used to cater for the unseen scenarios when attempting to close a file in Python.

Example

Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

Try:
file_name=open(“my text.txt”,encoding=’utf-8’)
finally:
file_name.close()

With this second approach, the user is assured that the file will close properly even if an unexpected error occurs interrupting the program flow.
In this case, the close() method does not have to be explicitly invoked as it is done internally.

Writing to a File in Python

Three modes are used to write into a Python file and they are the ‘w’r for writing, ‘a’ for appending, or ‘x’ for exclusive creation. The ‘w’ will overwrite the file and care should be exercised.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
file_name=open("my text.txt",'w',encoding = 'utf-8')
  file_name.write("In my first file\n")
  file_name.write("Great file\n\n")
  file_name.write("Has three lines\n")
```

Reading a file in Python

In Python, a file has to be opened in reading mode to read it. Several methods

are available to read a file in Python. If the size parameter is not stipulated, the method will read and return up to the end of the file.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
file_name=open("my text.txt",'r'. encoding='utf-8')
file_name.read(5)   # reads the first 5 data

file_name.read(5)   # reads the next 5 data

file_name.read()    # reads till end of file

file_name.read()    # further reading will empty sting
```

Exercise

a. Locate and open a .txt file on your computer using the open() in read mode.
b. In a. read the first three data of the file
c. Read the first five data of the file in a.
d. Read the entire file.
e. Continue reading after reading in d.

Using the for loop when reading a file in Python

The for loop allows us to read a file line after line.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
file_name=open ("my text.txt",'w', encoding = 'utf-8')
  file_name.write("In my first file\n")
  file_name.write("Great file\n\n")
  file_name.write("Has three lines\n")
```

We now include the control statement.

```
for line in file_name:
  print(line, end = '')
```

Note

The file has a newline character '\n'. the print() last parameter helps to avoid the creation of two newlines when outputting.

Readline() for Reading Individual File Lines in Python

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
file_name.readline()
file_name.readline()
file_name.readline()
file_name.readline()
```

Readlines() Python Method

For the readlines() method, it will scan the entire file lines. After reading the entire file, the reading method will report empty status. The reason for empty status is because there are no more arguments for the method to process. The read() will go through each line in the file until there are no more lines to scan.

Example

Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
file_name=open("my text.txt"')
  file_name.write("In my first file\n")
  file_name.write("Great file\n\n")
  file_name.write("Has three lines\n")
file_name.readlines()
```

Exercise

a. Use the readline() method to read each line of a .txt on your computer.
b. Now use the readlines()method to read the entire contents of the .txt in a.

File Methods in Python

These methods enable the user to manipulate files in an easy and efficient manner. The methods are preloaded and the user only needs to understand where they can apply.

Method	Description		Method	Description
detach()	Separate the underlying binary buffer from the TextIOBase and return it.		flush()	Flush the write buffer of the file stream.
read(n)	Read at most n characters from the file. Reads till the end of file if it is negative or None.		fileno()	Return an integer number (file descriptor) of the file.
isatty()	Return True if the file stream is interactive.		readable()	Returns True if the file stream can be read from.
readline(n=-1)	Read and return one line from the file. Reads in at most n bytes if specified.		seek(offset,from=SEEK_SET)	Change the file position to offset bytes, in reference to from (start, current, end).
close()	Close an open file. It has no effect if the file is already closed.		readlines(n=-1)	Read and return a list of lines from the file. Reads in at most n bytes/characters if specified.
writable()	Returns True if the file stream can be written to.		tell()	Returns the current file location.
seekable()	Returns True if the file stream supports random access.		truncate(size=None)	Resize the file stream to size bytes. If size is not specified, resize to the current location.
write(s)	Write string s to the file and return the number of characters written.		writelines(lines)	Write a list of lines to the file.

Directory in Python

Python allows us to arrange several files into different directories for easier handling. The collection of files and subdirectories in Python is known as a directory. The os module in Python contains methods for working with directories.

Getting Current Directory in Python

The keyword getcwd() method is used to get the current working directory. The method will return the current working directory in a string form. The getwcwdb()is also used to get a directory.

Changing Directory

The chdir() method helps modify the existing directory. However, the new path

we intend to create should be given a string as a method. The path elements can be separated using the backward slash\ or the forward slash/.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
os.chdir('C:\\Tutorial')

print(os.getcwd())
```

Exercise
Use the chdir() to change a directory in the Python root folder on your computer.
Files and List Directories
The listdir() method in Python is used to determine all files and subdirectories within a directory.
The listdir() method accepts a path and gives file lists and subdirectories in that particular path. The listdir() will return from the current working directory if no path is specified.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
print(os.getcwd())
C:\Tutorial
os.listdir()
os.listdir('D:\\')
```

Creating a New Directory

The mkdir() method can be used to create a new directory. The method accepts the path of the new directory and will create a new directory in the current working directory in case the particular path is not defined.

os.mkdir('week2')

os.listdir()

File or a Directory Renaming in Python

In Python, the rename()method is used to rename a file or a directory. The old name is given as argument 1 and the new name is given as argument 2.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
os.listdir()
os.rename('week2','fresh_name')
os.listdir(
os.listdir()
```

Exercise

Create a directory using the Python method and name it Lesson. Rename it using a Python method to Python Lessons.

Removing File or Directory in Python

The remove() method is used to delete a file in Python. Likewise, rmdir() is used to remove an empty directory.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

os.listdir()
['mine_direct']

os.rmdir('mine_direct')

Errors and Exceptions

When the Python interpreter encounters errors it will raise exceptions. For instance, dividing a number by zero will lead to an exception.

Example of an Error
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
if y < 3
   if y < 3
```

at runtime errors can still occur like when we open a file that does not exist. A file may not exist because it has been renamed and we are accessing using the old name or the file has been deleted. The file could have the same name but has changed the file extension. Python will create an exception object whenever these runtime error types occur.

Inbuilt Exceptions in Python

Python has several inbuilt exceptions that are flagged when associated errors arise. The local() method can help list all inbuilt exceptions in Python.

Exception	Source of Error	Exception	Source of Error
EOFError	Raised when the input() functions hit end-of-file condition.	AttributeError	Raised when attribute assignment or reference fails.
GeneratorExit	Raise when a generator's close() method is called.	ImportError	Raised when the imported module is not found.
KeyError	Raised when a key is not found in a dictionary.	IndexError	Raised when the index of a sequence is out of range.
MemoryError	Raised when an operation runs out of memory.	KeyboardInterrupt	Raised when the user hits interrupt key.
AssertionError	Raised when assert statement fails.	FloatingPointError	Raised when a floating point operation fails.
OSError	Raised when system operation causes system related error.	OverflowError	Raised when the result of an arithmetic operation is too large to be represented.
ReferenceError	Raised when a weak reference proxy is used to access a garbage collected referent.	NotImplementedError	Raised by abstract methods.
NameError	Raised when a variable is not found in the local or global scope.	RuntimeError	Raised when an error does not fall under any other category.

Exception Handling in Python

If exceptions occur, the execution of a program stops and the exception is passed to the calling process for handling. If the exception is not handled, a program will crash. Thought: Have you tried opening a smartphone app that says “the app has stopped”. Can you imagine a program that crashes if it cannot file the image you trying to attach? Exception handling is critical in creating effective programs and also for security. Can you imagine if a hacker realizes that your program will crash and require restarting each time it fails to load an image, the hacker will have a leeway disrupting your software? Fortunately, exception handling allows us to anticipate and give way forward to a program should it encounter an exception.

Handling an Exception by Catching the Exception in Python

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import sys
my_list= ['b', 1 ,3]
for entry in my_list:
   try:
      print("Our entry is", my_entry)
      r = 1/int(my_entry)
      break
   except:
      print("Unfortunate",sys.exc_info()[0],"has occured.")
      print("Try again.")
      print()
print("The reciprocal of the number",my_entry,"is",r)
import sys
```

User-defined Exception

Sometimes a user may need to create own exceptions that best align with your programming needs and these are known as user-defined exceptions.
A user has to make a new class and derive from the class, Exception class.

Summary

Files are used for future storage. To read from a file we need to open it and once through with it, we have to close it to free the resources tied with the file in Python. The inbuilt function open() in Python is used to launch a file. When this function is invoked, a file object is accessed and loaded sometimes referred to as a handle and it is used to modify or read the file accordingly. For the readlines() method, it will scan the entire file lines. After reading the entire file, the reading method will report empty status. The reason for empty status is because there are no more arguments for the method to process. The read() will go through each line in the file until there are no more lines to scan.

The ‘r’ is used to launch the file for reading and is the default. The ‘w’ is preferred when launching a file for writing. For exclusive file creation, we use

the ‘x’ mode. The operation will be unsuccessful if the file already we want already exists. The other mode, ‘a’ is used to launch a file for adding data at the file end while retaining earlier content. The ‘a’ model will create a new file in case it does not exist. For the ‘t’ mode, it will launch the file in default text mode. The ‘b’ mode is used to launch the file in binary mode. Lastly, the ‘+’ mode is used for launching the file to allow reading and writing.

Chapter 2: Main Concepts of Machine Learning

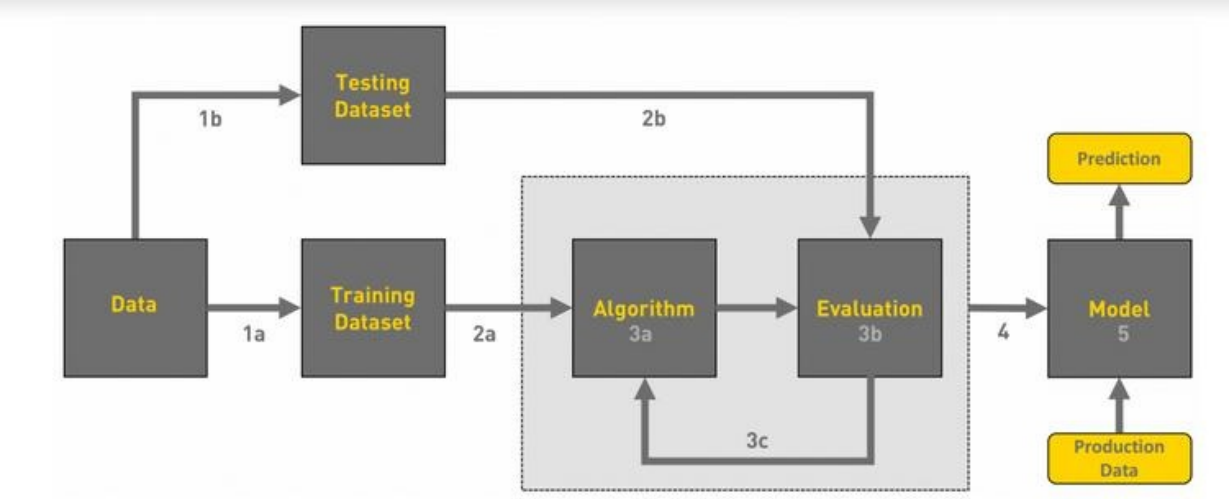
What is a ML workflow? What are the core steps of a typical ML workflow?

This chapter discusses the workflow of a ML project this includes all the steps required to build the proper ML project from scratch.

We will also go over data pre-processing, data cleaning, feature exploration and feature engineering and show the impact that it has on ML Model Performance. We will also cover a couple of the pre-modeling steps that can help to improve the model performance.

Python Libraries that would need to achieve the task:

- Numpy
- Pandas
- Sci-kit Learn
- Matplotlib



Overview of the Workflow of ML

Overview of the Workflow of ML

Understanding the ML workflow

We can define the ML workflow in 3 stages.

1. Gathering data
2. Data pre-processing
3. Researching the model that will be best for the type of data
4. Training and testing the model
5. Evaluation

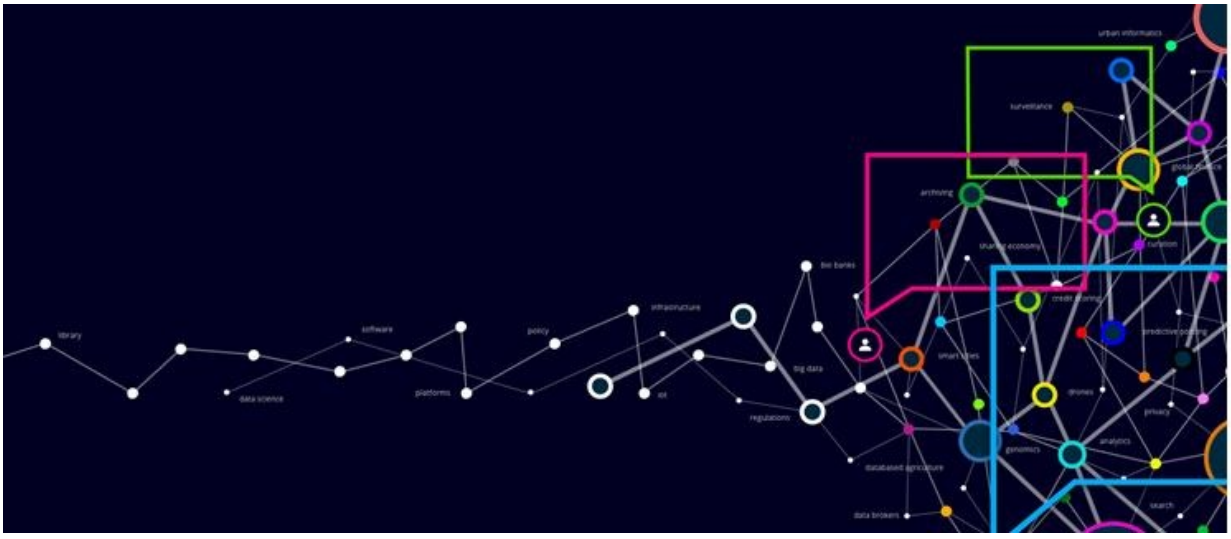
Okay but first let’s start from the basics

What is the ML Model?

The ML model can be defined as a piece of program which a data scientist or an engineer makes it smart by training it using data. So, for example, if you feed the model with garbage it will produce garbage meaning that the trained model provides wrong or false predictions if fed with invalid data or garbage.

- ## Data pre-processing

It is one of the essential procedures in ML. It is the most important step that helps in making ML models more accurately. In ML, there is an 80/20 rule. Every data scientist should spend 80% time for data pre-processing and 20% time to perform the analysis.



What is data pre-processing?

Data pre-processing refers to a process of cleaning the raw data. The data which is collected in the real world is usually converted to a clean set of data. Whenever the data is collected from varying sources, it is usually in a raw form, and this data is not feasible for the analysis phase.
Consequently, specific procedures are followed to turn the data into a reduced clean set of data; this process is known as pre-processing of data.

- **Researching the model that will be best for the type of data**

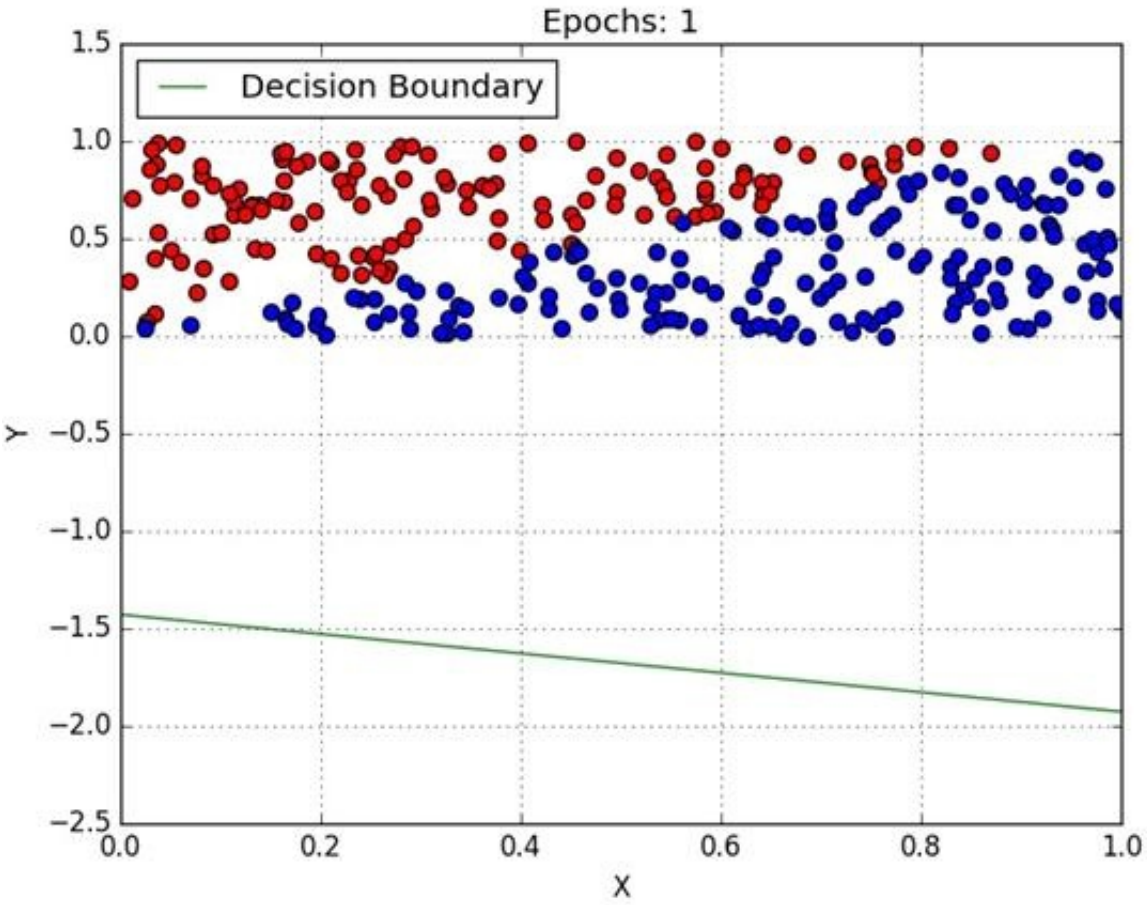
Our main goal is to train the best performing model possible, using the pre-processed data.

Supervised Learning:

In Supervised learning, an Artificial Intelligent system is presented with labeled information or data, which means that each data tagged with the correct label.
The supervised learning is categorized into two other categories which are **Regression and Classification**

Classification

The classification problem is when the target variable is *categorical* (meaning that the output could be classified-it belongs to either Class A or B or something else).
A classification problem refers to when an output variable is a group, such as *blue* or *red*, *no disease* or *disease* or *not spam* or *spam*.

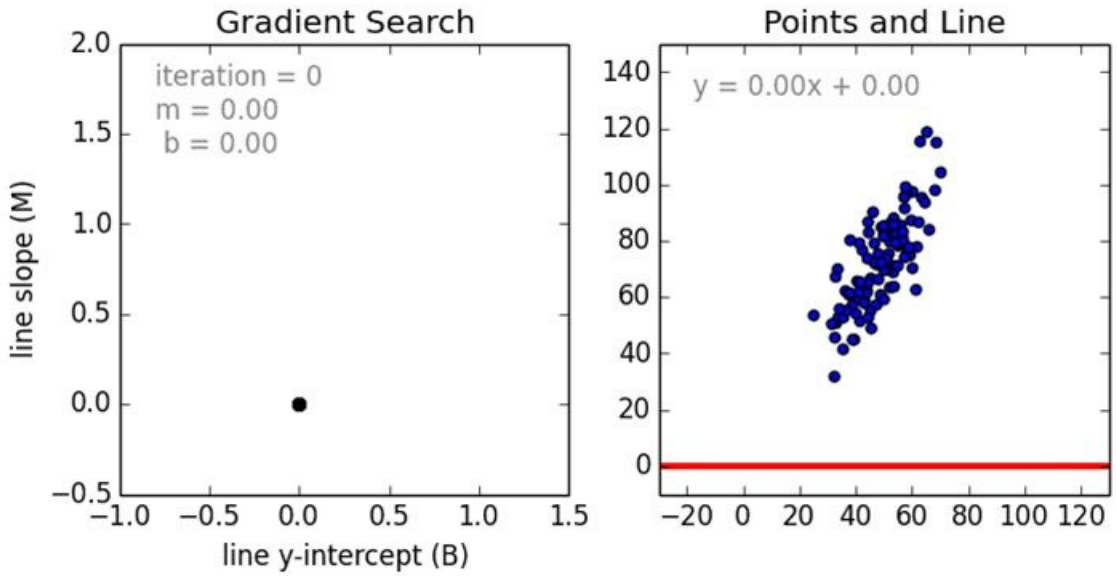


These some most used classification algorithms.

- KNN
- Naive Bayes
- Random Forest
- Logistic

Regression:

A regression problem happens when the targeted variable is continuous (meaning that the output is numeric).



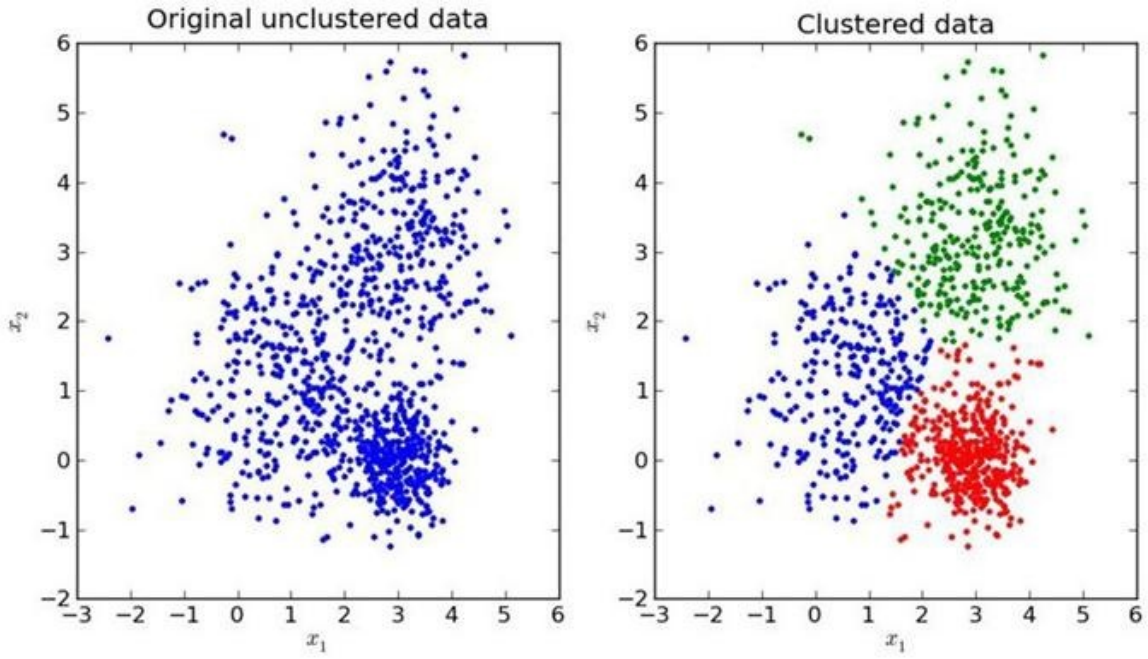
Regression

As shown in the above representation, we can imagine that the graph's X-axis is the 'Test scores' and the Y-axis represents 'IQ.' So we try to create the best fit line in the given graph so that we can use that line to predict any approximate IQ that is not present in the given data.

These some most used regression algorithms.

- Linear
- Support Vector
- Decision Trees
- Gaussian Progresses
- Ensemble Methods

Unsupervised Learning:

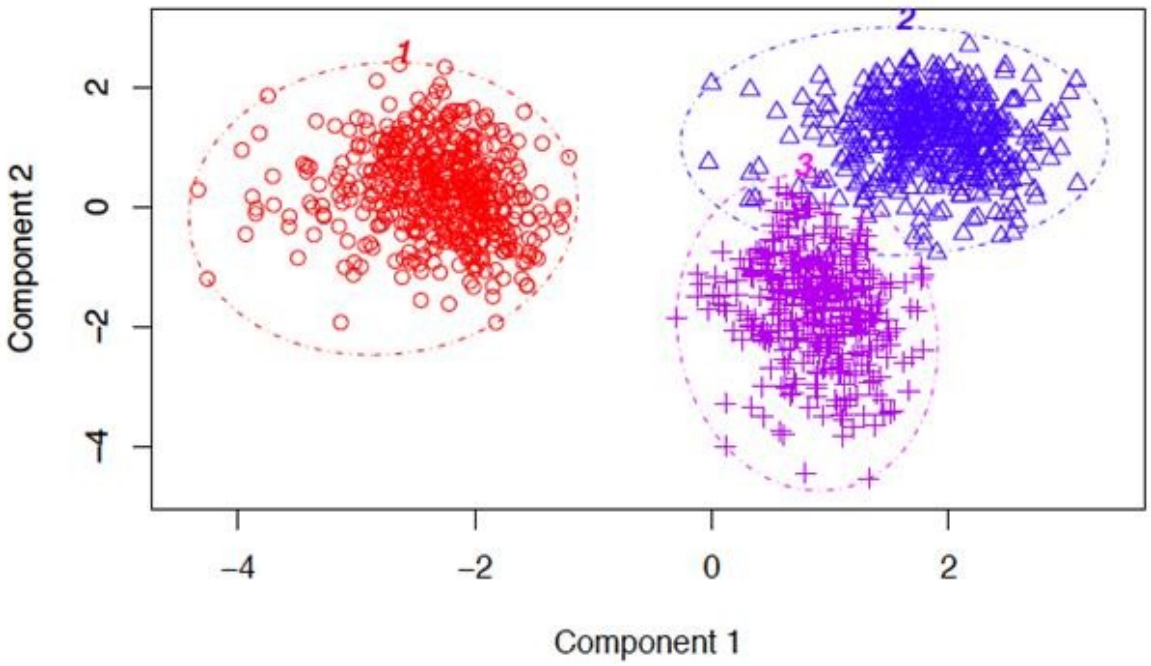


For unsupervised learning an Artificial Intelligence system is fed with un-categorized and unlabeled data and the algorithms of the system process the data before any training. The resultant output depends on the programmed or coded algorithms. Exposing a system to unsupervised learning is one of the ways of testing Artificial Intelligence.

The unsupervised learning is divided into two other groups known as Clustering and Association.

Clustering:

A set of inputs is divided into groups. As opposed to classification, the groups are not known at the beginning, making this an unsupervised process or task.

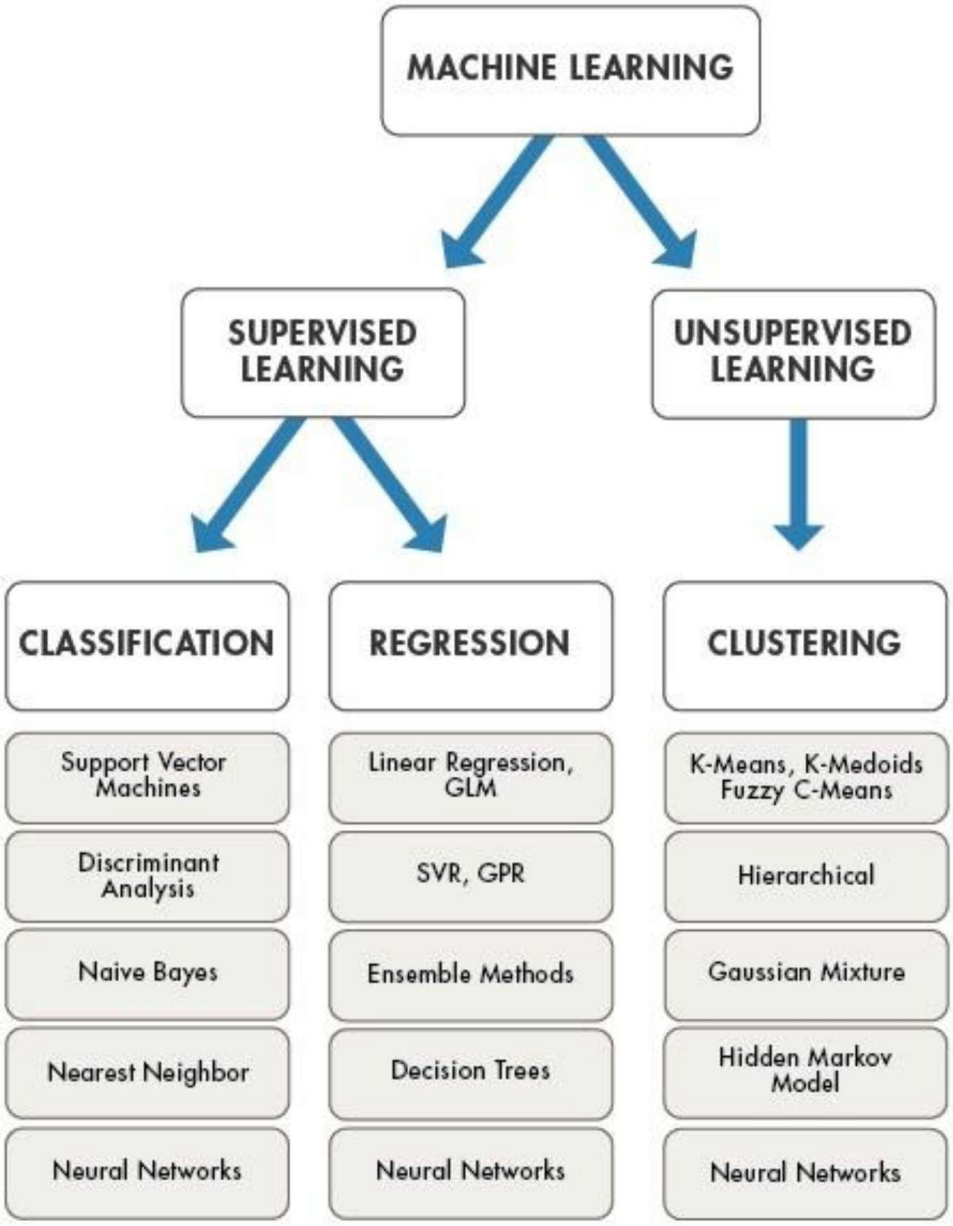


Clustering

Methods used for clustering are:

- K-Means
- Gaussian
- Boosting
- Hierarchical
- Spectral

Overview of models under categories:



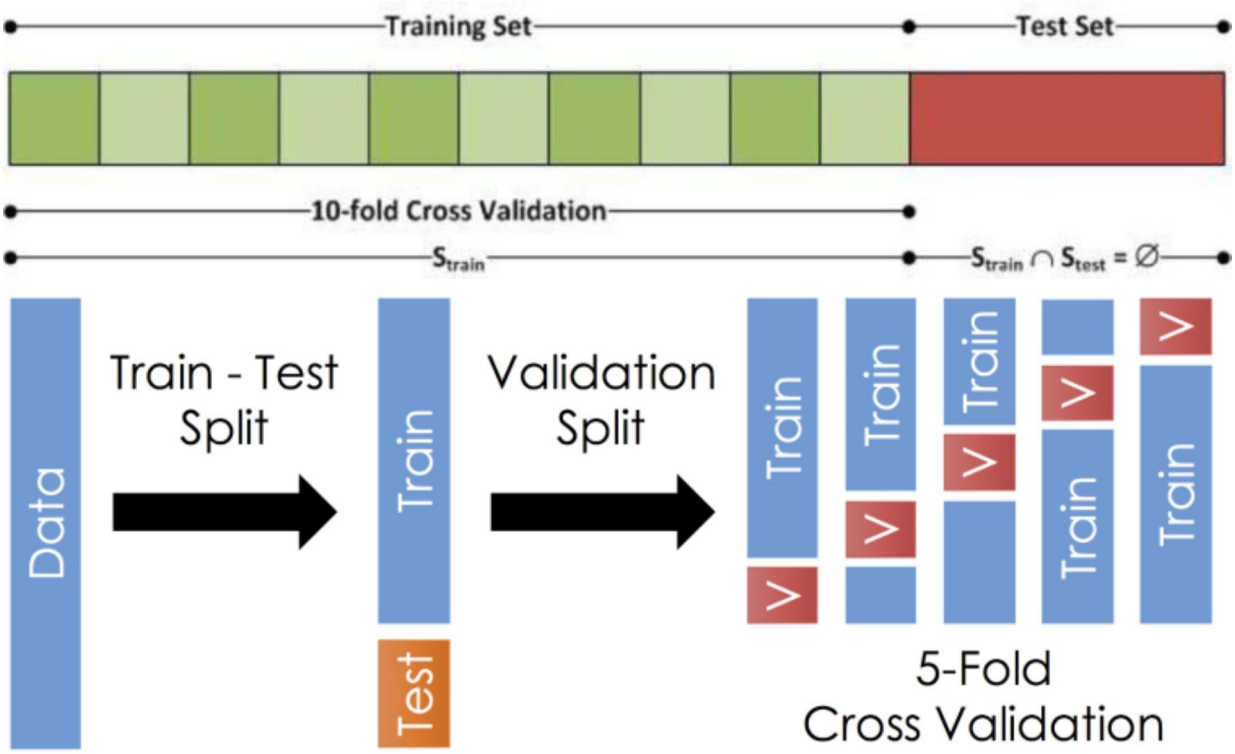
Overview of models

- **Training and testing the model on data**

For training a model we initially split the model into 3 three sections which are **training data**, **validation data** and **testing data**.

The classifier is usually trained using the ***training set of data***, make sure you prepare the needed parameters using the ***validation set*** of data and then test the functioning of the classifier using an unseen test set of data. An essential thing to note is that when you are training the classifier only the training and validation

set of data is available. The test set of data should not be used when training the classifier. The test set is normally available when testing the classifier.



If I had to summarize the main highlights of ML advances in 2018 in a few headlines, these are the ones that I would probably come up with:

- AI hype and fear mongering cool down.
- More focus on concrete issues like fairness, interpretability, or causality.
- Deep learning is here to stay and is useful in practice for more than image classification (particularly for NLP).
- The battle on the AI frameworks front is heating up, and if you want to be someone you better publish a few frameworks of your own.

If the year 2017 was probably the cusp of Artificial Intelligence fear mongering and hype (as I mentioned in last year's answer), 2018 seems to have been the year where we have started to all cool down a bit. While it is true that some figures have continued to push their message of AI fear, they have probably been too busy with other issues to make of this an important point of their agenda. At the same time, it seems like the press and others have come to peace with the

idea that while self-driving cars and similar technologies are coming our way, they won't happen tomorrow. That being said, there are still voices defending a bad idea that we should regulate AI instead of focusing on regulating its outcomes.

It is good to see that this year though, the focus seems to have shifted to more concrete issues that can be addressed. For example, there has been a lot of talk around **fairness,** and there are not only several conferences on the topic (see FATML or ACM FAT) even some online courses like this one by Google.

Generation Unlimited

Along these lines, other issues that have been greatly discussed this year include ***interpretability***, ***explanations***, and ***causality***. Starting with the latter, causality seems to have made it back to the spotlight mostly because of the publication of Judea Pearl's "The Book of Why." Not only did the author decided to write his first "generally accessible" book, but he also took to twitter to popularize discussions around causality. Even the popular press has written about this as being a "challenge" to existing Artificial Intelligence approaches (see this article in The Atlantic, for example). Even the best paper award at the ACM Recsys conference went to a paper that addressed the issue of how to include causality in embedding. That being said, many other authors have argued that causality is somewhat of a theoretical distraction, and we should focus again on more concrete issues like interpretability or explanations. Speaking of explanations, one of the highlights in this area might be the publication of the paper and code for Anchor, a follow up to the well-known LIME model by the same authors.

While there are still questions about the Deep Learning as the most general AI paradigm (count me in with those raising questions), while we continue to skim over the nth iteration of the discussion about this between Yann LeCun and Gary Marcus, it is clear that Deep Learning is not only here to stay, but it is still far from having reached a plateau in terms of what it can deliver. More concretely,

during this year Deep Learning approaches have shown unprecedented success in fields different from Vision, ranging from Language to Healthcare.

It is probably in the area of **NLP**, where we have seen the most interesting advances this year. If I had to choose the most impressive AI applications of the year, both of them would be NLP (and both come from Google). The first one is Google's super useful smart compose, and the second one is their Duplex dialog system.

A lot of those advances have been accelerated by the idea of using **language models**, popularized this year. We have then seen other (and improved) approaches like Allen's ELMO, Open AI's transformers, or, more recently Google's BERT, which beat many SOTA results out of the gate. These models have been described as the "Imagenet moment for NLP" since they show the practicality of transfer learning in the language domain by providing ready-to-use pre-trained and general models that can also be fine-tuned for specific tasks. Besides language models, there have been plenty of other interesting advances like Facebooks multilingual embedding, to name another one. It is interesting to note that we have also seen how quickly these and other approaches have been integrated into more general NLP frameworks such as AllenNLP's or Zalando's FLAIR.

Speaking of **frameworks**, this year the "war of the AI frameworks" has heated up. Surprisingly, **Pytorch** seems to be catching up to **TensorFlow** just as Pytorch 1.0 was announced. While the situation around using Pytorch in production is still sub-optimal, it seems like Pytorch is catching up on that front faster than Tensor Flow is catching up on usability, documentation, and education. Interestingly it is likely that the choice of Pytorch as the framework on which to implement the Fast.ai library has played a big role. That being said, Google is aware of all of this and is pushing in the right direction with the inclusion of Keras as a first-class citizen in the framework or the addition of key developer-focused leaders like Paige Bailey. In the end, we all benefit from having access to all these great resources, so keep them coming!

Interestingly, another area that has seen a lot of interesting developments in the framework space is **reinforcement learning**.

While I don't think RL research advances have been as impressive as in previous years (only the recent Impala work by DeepMind comes to mind), it is surprising to see that in a single year we have seen all major Artificial Intelligence players publish an RL Framework. Google published the Dopamine framework for research while Deepmind (also inside of Google) published the somewhat competing TRFL framework. Facebook could not stay behind and published Horizon while Microsoft published TextWorld, which is more specialized for text-based training agents. Hopefully, all of this open source goodness will help us see a lot of RL advances in 2019.

Just to finish up on the front of the framework, I was happy to see that Google recently published TFRank on top of Tensor Flow. The ranking is an extremely important ML application that is probably getting less love than it deserves lately.

It might seem like Deep learning has ultimately removed the need to be smart about your data, but that is far from true.

There are still very interesting advances in the field that revolve about the idea of improving data. For example, while **data augmentation** has been around for some time and is vital for many DL applications, this year Google published auto-augment, a deep reinforcement learning approach to automatically augment training data. An even more extreme idea is to train DL models with **synthetic data**. This has been tried in practice for some time and is seen as key to the future of Artificial Intelligence by many.

NVidia presented interesting novel ideas in their Training Deep Learning with Synthetic Data paper. In our "Learning from the Experts," we also showed how to use expert systems to generate synthetic data that can then be used to train DL systems even after combining with real-world data. Finally, also interesting is the approach of reducing the need to have large quantities of hand-labeled data by using ***weak supervision***. Snorkel is a very interesting project that aims at

facilitating this approach by providing a generic framework.

As far as more ***foundational breakthroughs*** in AI, it might be me and my focus, but I haven't seen many. I don't entirely agree with Hinton when he says that this lack of innovation is due to the field having "a few senior guys and a gazillion young guys" although it is true that there is a trend in science where breakthrough research is done at a later age.

In my opinion, the main reason for the current lack of breakthroughs is that there are still many interesting practical applications of existing approaches and variations so it is hard to risk in approaches that might not be practical right away. That is even more relevant when most of the research in the field is sponsored by large companies. While being highly empirical and using known approaches, it opens the door to uncovering new ones since it proves that the one that is usually regarded as optimal is not. Another highly exploratory paper is the recent NeurIPS best paper award winner ***Neural Ordinary Differential Equations*** which challenges a few fundamental things in DL including the notion of layers itself.

Interestingly, this last paper was motivated by a project where the authors were looking into ***healthcare*** data (more concretely Electronic Health Records). I cannot finish this summary without referring to the area of research in the intersection of AI and Healthcare since that is where my focus at Curai is at. Unfortunately, so much is going on in this space that I would need another post only for this. So, I will only point you to the papers that were published at the MLHC conference and the ML4H NeurIPS workshop. Our team at Curai managed to get papers accepted at both, so you will find our papers there among many other interesting ones that should give you an idea of what is going on in our world.

Object and Class in Machine Learning using Python

Python supports different programming approaches as it is a multi-paradigm. An object in Python has an attribute and behavior. It is essential to understand

objects and classes when studying machine learning using Python object-oriented programming language.

Example

Car as an object:

Attributes: color, mileage, model, age

Behavior: reverse, speed, turn, roll, stop, start.

Class

It is a template for creating an object.

Example

class Car:

Important:

By convention, we write the class name with the first letter as uppercase. A class name is in singular form by convention.

Syntax

class Name_of_Class:

From a class, we can construct objects by simply making an instance of the class. The class_name() operator creates an object by assigning the object to the empty method.

Object/Class Instantiation

From our class Car, we can have several objects such as a first car, second care or SUVs.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
my_car=Car()
  pass
```

Exercise

a. Create a class and an object for students.
b. Create a class and an object for the hospital.
c. Create a class and an object for a bank.
d. Create a class and an object for a police department.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Car:
category=”Personal Automobile”
   def __init__(self, model, insurance):
      self.model = model
      self.insurance =insurance
subaru=Car(“Subaru”,”Insured”)
toyota=Car(“Toyota”,”Uninsured”)
print(“Subaru is a {}”.format(subaru._class_.car))
print(“Toyota is a {}”.format(toyota._class_.car))
print(“{} is {}”.format(subaru.model, subaru.insurance))
print(“{} is {}”.format(toyota.model, toyota.insurance))
```

Methods

Functions defined within a body of the class are known as methods and are basic functions. Methods define the behaviors of an object.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
def __init__(self, model, insurance):
        self.model = model
        self.insurance =insurance
    def ignite(self, ignite):
        return "{} ignites {}".format(self.model, ignition)
    def stop(self):
        return "{} is now stopping".format(self.model)
subaru=Car(“Subaru”,”Insured”)
print(subaru.ignite("'Fast"'))
print(subaru.stop())
```

Important
The methods ignite() and stop() are referred to as instance methods because they are an instance of the object created.

Exercise
a. Create a class Dog and instantiate it.
b. Create a Python program to show the names of two dogs and their two attributes from a.

Inheritance in Machine Learning using Python

A way of creating a new class by using details of existing class devoid of

modifying it is called inheritance. The derived class or child class is the newly formed class while the existing class is called a parent or base class.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Dog:
    def __init__(self):
        print("Dog is available")
    def whoisThis(self):
        print("Dog")
    def walk(self):
        print("Walks gently")
class Spitz(Dog):       #Child class
    def __init__(self):
        super().__init__()
        print("Spitz is now available")
    def whoisThis(self):
        print("Pitbull")
    def wag(self):
        print("Strong")
pitbull = Pitbull()
pitbull.whoisThis()
pitbull.walk()
pitbull.wag()
```

Explanation

We created two Python classes in the program above. The classes were Dog as the base class and Pitbull as the derived class. The derived class inherits the functions of the base class. The method _init_() and the function super() are used to pull the content of _init_() method from the base class into the derived class.

Encapsulation in Machine Learning using Python

Encapsulation in Python Object Oriented Programming approach is meant to help prevent data from direct modification. Private attributes in Python are denoted using a single or double underscore as a prefix.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

“__” or “_”.

```
class Tv:
    def __init__(self):
        self.__Finalprice = 800
    def offer(self):
        print("Offering Price: {}".format(self.__finalprice))
    def set_final_price(self, offer):
        self.__finalprice = offer
t = Tv()
t.offer()
t.__finalprice = 950
t.offer()
# using setter function
t.setFinalPrice(990)
t.sell()
```

Explanation

The program defined a class Tv and used _init_(0 methods to hold the final offering price of the TV. Along the way, we attempted to change the price but

could not manage. The reason for the inability to change is because Python treated the _finalprice as private attributes. The only way to modify this value was through using a setter function, setMaxPrice() that takes price as a parameter.

Polymorphism

In Python, polymorphism refers to the ability to use a shared interface for several data types.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Tilapia:
    def swim(self):
        print("Tilapia can swim")
    def fly(self):
        print("Tilapia cannot fly")
class Shark:
    def jump(self):
        print("Shark can't fly")
    def swim(self):
        print("Shark can swim")
def jumping_test(fish):
    fish.jump()
bonny = Tilapia()
biggy = Shark()
jumping_test(bonny)
jumping_test(biggy)
```

Explanation

The program above has defined two classes Tilapia and Shark all of which share the method jump() even though they have different functions. By creating common interface jumping_test() we allowed polymorphism in the program

above. We then passed objects bonny and biggy in the jumping_test() function.

Exercise

a. In a doctor consultation room suggest the class and objects in a programming context.
b. In a football team, suggest programming class and objects.
c. In a grocery store, suggest programming class and objects.

Class Definition in Python

The keyword def is used to define a class in Python. The first string in a Python class is used to describe the class even though it is not always needed.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
class Dog
‘’’Briefly taking about class Dog using this docstring’’’
   Pass
```

Example 2
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
Class Bright:
“My other class”
b=10
def salute(self):
```

```
print(‘Welcome’)
print(Bright.b)

print(Bright.salute)
print(Bright.__doc__)
```

Object Creation in Python

Example from the previous class
Open the previous program file with class Bright

```
student1=Bright()
```

Explanation
The last program will create object student1, a new instance. The attributes of objects can be accessed via the specific object name prefix. The attributes can be a method or data including the matching class functions. In other terms, Bright.salute is a function object and student1.salute will be a method object.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
class Bright:
    "Another class again!"
    c = 20
    def salute(self):
        print('Hello')
student2 = Bright()
print(Bright.salute)
```

```
print(student2.salute)
student2.salute()
```

Explanation

We invoked the student2.salute() despite the parameter ‘self’ and it still worked without placing arguments. The reason for this phenomenon is because each time an object calls its method, the object itself is passed as the first argument. The implication is that student2.salute() translates into student2.salute(student2). It is the reason for the ‘self; name.

Constructors

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class NumberComplex
class ComplexNumber:
    def __init__(self,realnum = 0,i = 0):
        self.real = realnum
        self.imaginarynum = i
    def getData(self):
        print("{0}+{1}j".format(self.realnumber,self.imaginarynum))
complex1 = NumberComplex(2,3)
complex1.getData()
complex2 = NumberComplex(5)
complex2.attribute = 10
print((complex2.realnumber, complex2.imaginarynumber, complex2.attribute))
complex1.attribute
```

Deleting Objects and Attributes

The del statement is used to delete attributes of an object at any instance.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
complex1 = NumberComplex(2,3)
del complex1.imaginarynumber
complex1.getData()
del NumberComplex.getData
complex1.getData()
```

Deleting the Entire Object

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
complex1=NumberComplex(1,3)
del complex1
```

Explanation
When complex1=NumberComplex(1,3) is done, a new instance of the object gets generated in memory and the name complex1 ties with it. The object does not immediately get destroyed as it temporarily stays in memory before the garbage collector purges it from memory. The purging of the object helps free resources bound to the object and enhances system efficiency. Garbage destruction Python refers to automatic destruction of unreferenced objects.

Inheritance in Python

In Python inheritance allows us to specify a class that takes all the functionality from the base class and adds more. It is a powerful feature of OOP.

Syntax

```
class ParentClass:
  Body of parent class
class ChildClass(ParentClass):
  Body of derived class
```

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Rect_mine(Rect_mine):
    def __init__(self):
        Shape.__init__(self,4)
    def getArea(self):
        s1, s2, s3,s4 = self.count_sides
        perimeter = (s1+s2+s3+s4)
        area = (s1*s2)
        print('The rectangle area is:' %area)
```

Example 2

```
r = rect_mine()
r.inputSides()
Type b1 : 4
Type l1 : 8
Type b2 : 4
```

Type l1: 8

r.dispSides()
Type b1 is 4.0
Type l1 is 8.0
Type b2 is 4.0
Type l1 is 8.0
r.getArea()

Method Overriding in Python

When a method is defined in both the base class and the derived class, the method in the child class/derived class will override the parent/base class. In the above example, the _init_() method in Rectangle class will override the _init_() in Shape class.

Inheritance in Multiple Form

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
class Parent1:
   pass
class Parent2:
   pass
class MultiInherit(Parent1, Parent2):
   pass
```

In this case, MultiInherit is derived from class Parent1 and Parent2.
Multilevel Inheritance
Inheriting from a derived class is called multilevel inheritance.

Example

Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
class Parent:
    pass
class Multilevel1(Parent):
    pass
class Multilevel2(Multilevel1):
    pass
```

Explanation
Multilevel1 derives from Parent, and Multilevel2 derives from Multilevel1.
Method Resolution Order

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
print(issubclass(list,object))
print(isinstance(6.7,object))
print(isinstance("Welcome",object))
```

Explanation
The particular attribute in a class will be scanned first. The search will continue into parent classes. This search does not repeat searching for the same class twice. The approach or order of searching is sometimes called linearization of multi derived class in Python. The Method Resolution Order refers to the rules needed to determine this order.

Operator Overloading in Machine Learning

Inbuilt classes can use operators and the same operators will behave differently with different types. An example is the + that depending on context will perform concatenation of two strings, arithmetic addition on numbers, or merge lists. Operating overloading is an OOP feature that allows assigning varying meaning to an operator subject to context.

Making Class Compatible with Inbuilt Special Functions

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Planar:
    def __init__(self, x_axis= 0, y_axis = 0):
        self.x_axis = x_axis
        self.y_axis = y_axis
    def __str__(self):
        return "({0},{1})".format(self.x_axis,self.y_axis)
```

Explanation

```
planar1=Planar(3,5)
print(planar1)          #The output will be (3,5)
```

Using more inbuilt methods

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Planar:
    def __init__(self, x_axis= 0, y_axis = 0):
        self.x_axis = x_axis
        self.y_axis = y_axis
str(planar1)
format(planar1)
```

Explanation

It then follows that each time we invoke format(planar1) or str(planar1), Python is in effect executing planar1._str_() thus the name, special functions.

Operator + Overloading

The _add_() function addition in a class will overload the +.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
class Planar:
    def __init__(self, x_axis= 0, y_axis = 0):
        self.x_axis = x_axis
        self.y_axis = y_axis
    def __str__(self):
        return "({0},{1})".format(self.x_axis,self.y_axis)
    def __add__(self,z):
        x_axis = self.x_axis + z.x_axis
        y_axis = self.y_axis + z.y_axis
        return Planar(x_axis,y_axis)
```

Exercise

a. Print planar1 + planar2 from the example above.

Explanation

When you perform planar1+planar2 in Python, it will call planar._add_(planar2) and in turn Planar._add_(planar1, planar2).

Revisit Logical and Comparison Operators

Exercise

a. Given x=8, y=9, write a Python program that uses logical equals to test if x is equal to y.

b. Write a program that evaluates x!=y in Python programming language.

c. Write and run the following program

```
m = True
n = False
print('m and n is',m and n)
print('m or n is',m or n)
print('not m is',not n)
```

d. From the program in c., which program statement(s) evaluates to True, or False.

e. Write and run the following program in Python

```
m1 = 15
n1 = 15
m2 = 'Welcome'
n2 = 'Welcome'
m3 = [11,12,13]
n3 = [11,12,13]
print(m1 is not n1)
print(m2 is n2)
print(m3 is n3)
```

f. Which program statement(s) generate True or False states in e.

g. Write and run the following program

```
m = 'Welcome'
n = {11:'b',12:'c'}
print('W' in m)
print('Welcome' not in m)
print(10 in n)
print('b' in n)
```

h. Which program statement(s) in g. return True or False states.

The special functions needed for overloading other operators are listed below.

Operator	Expression	Internally
Subtraction	p1 - p2	p1.__sub__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Remainder	p1 % p2	p1.__mod__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Addition	p1 + p2	p1.__add__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)

Comparison Operators Overloading

In Python, comparison operators can be overloaded.

Example

```
class Planar:
    def __init__(self, x_axis= 0, y_axis = 0):
        self.x_axis = x_axis
        self.y_axis = y_axis
    def __str__(self):
        return "({0},{1})".format(x_axis,y_axis)
    def __lt__(self,z):
        self_magnitude = (x_axis ** 3) + (y _axis ** 3)
        z_magnitude = (z.x_axis ** 3) + (z.y_axis ** 3)
        return self_magnitude < z_magnitude
```

Exercise

a. Perform the following to the example above Planar(1,1)
b. Again perform Planar(1,1) in the above example.
c. Finally, perform Planar(1,1) from the above example.

Summary of Special Functions for Implementing Comparison Operator Overloading

Operator	Expression	Internally
Subtraction	p1 - p2	p1.__sub__(p2)
Power	p1 ** p2	p1.__pow__(p2)
Remainder	p1 % p2	p1.__mod__(p2)
Multiplication	p1 * p2	p1.__mul__(p2)
Division	p1 / p2	p1.__truediv__(p2)
Addition	p1 + p2	p1.__add__(p2)
Floor Division	p1 // p2	p1.__floordiv__(p2)
Bitwise Right Shift	p1 >> p2	p1.__rshift__(p2)
Bitwise OR	p1 | p2	p1.__or__(p2)
Bitwise AND	p1 & p2	p1.__and__(p2)
Bitwise XOR	p1 ^ p2	p1.__xor__(p2)
Bitwise NOT	~p1	p1.__invert__()
Bitwise Left Shift	p1 << p2	p1.__lshift__(p2)

Summary

Python supports different programming approaches as it is a multi-paradigm. An object in Python has an attribute and behavior. From a class, we can construct objects by simply making an instance of the class. The class_name() operator creates an object by assigning the object to the empty method. The keyword def is used to define a class in Python. The first string in a Python class is used to describe the class even though it is not always needed. When a method is defined in both the base class and the derived class, the method in the child class/derived class will override the parent/base class. In the above example, the

init() method in Rectangle class will override the _init_() in Shape class.

Inbuilt classes can use operators and the same operators will behave differently with different types. An example is the + that depending on context will perform concatenation of two strings, arithmetic addition on numbers, or merge lists. Operating overloading is an OOP feature that allows assigning varying meaning to an operator subject to context. From a class, we can construct objects by simply making an instance of the class. The class_name() operator creates an object by assigning the object to the empty method.
The _init_() function is a special function and gets called whenever a new object of the corresponding class is instantiated. Functions defined within a body of the class are known as methods and are basic functions. Methods define the behaviors of an object. In Python, polymorphism refers to the ability to use a shared interface for several data types. An illustration is a program that has defined two classes Tilapia and Shark all of which share the method jump() even though they have different functions. By creating common interface jumping_test() we allowed polymorphism in the program above. We then passed objects bonny and biggy in the jumping_test() function.

Chapter 3: Basics of Python

Python is both procedural and object-oriented coding language. It has a straightforward syntax. Python is cross-platform implying that it can be run on different Operating Systems environments such as Linux, Windows platform, Mac OS X platform, UNIX platform and can be ported to .NET and Java virtual machines. Python is free and open source. While most recent versions of Mac and Linux have Python preinstalled, it is recommended that one installs and runs the current version.

Installation of Python

Most recent versions of Linux and Mac have Python already installed in them. However, you might need to install Python, and the following are the steps for installing Python in Windows, Mac OS X or Linux.

Installing Python in Macintosh Operating System X

i. Visit Download Python page which is the credible site and click “Download Python 3.7.2 (The version may differ from the one stated here).

ii. When the download completes, click open the package and follow the instructions given. The installation should complete with “The installation was successful” prompt.

iii. Now, visit Download Notepad++ and download the text editor and install it by opening the package and following the message prompts. The Notepad++ text editor is free and suited to help write source code (raw text programming words).

Installation of Python in Linux Operating Systems

i. Install $ sudo apt-get install build-essential checkinstall

$ sudo apt-get install libreadline-gplv2-dev libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev libbz2-dev

ii. Visit Download Python page which is the official site and click "Download Python 3.7.2 (The version may differ from the one stated here).

iii. While in the terminal, go to the directory where the downloaded file is, and run the command: $ tar -xvf Python-3.7.2.tgz

Important: Use the appropriate file name depending on the version that you downloaded.

iv. Then go to the extracted directory and type

$ cd Python-3.7.2

v. It is now time to issue instructions to run the source code on your OS (Operating System)

Python Installation in Windows Operating System

i. Visit Download Python site which is the recommended site and click "Download Python 3.7.2 (The version may differ from the one stated here).

ii. When your download completes, open the package by clicking and follow the guidelines given. The Python installation should complete with "The installation was successful" prompt.

When you install Python successfully, it also installs a program known as IDLE along with it. IDLE is a graphical user interface when working with Python.

iii. Now, visit Download Notepad++ and download the text editor and install it by opening the package and following the message prompts. The Notepad++ text editor is free and suited to help write source code (raw text programming words).

Ways of Running Python

Now before we start running our first python program, it is important that we

understand how we can run python programs. Running or executing or deploying or firing a program simply means that we are making the computer process instructions/lines of codes. For instance, if the lines of codes (program) require the computer to display some message, then it should. The following are the ways or mode of running python programs. The interpreter is a special program that is installed when installing the Python package and helps convert text code into a language that the computer understands and can act on it (executing).

i. Immediate Mode

It is a way of running python programs that are not written in a file. We get into the immediate mode by typing the word python in the command line and which will trigger the interpreter to switch to immediate mode. The immediate mode allows typing of expressions directly, and pressing enter generates the output. The sign below is the Python prompt:

>>>

The python prompt instructs the interpreter to accept input from the user. For instance, typing 2+2 and pressing enter will display 4 as the output. In a way, this prompt can be used as a calculator. If you need to exit the immediate mode, type quit() or exit().

Now type 5 +3, and press enter, the output should be 8. The next mode is the Script Mode.

ii. Script Mode

The script mode is used to run a python program written in a file; the file is called a script.

Integrated Development Environment (IDE)

An IDE provides a convenient way of writing and running Python programs.

One can also use text editors to create a python script file instead of an IDE by writing lines of codes and saving the file with a .py extension. However, using an IDE can simplify the process of writing and running Python programs. The IDEL present in the Python package is an example of an IDE with a graphical user interface and gets installed along with the Python language. The advantages of IDE include helping getting rid of repetitive tasks and simplify coding for beginners. IDE provides syntax highlighting, code hinting, and syntax checking among other features. There also commercial IDE such as the PyScripter IDE that performs most of the mentioned functions.

Note:

We have presented what Python is, how to download and install Python, the immediate and script modes of Python IDE, and what is an IDE.

Your First Written Program in Python

The rest of the illustrations will assume you are running the python programs in a Windows environment.

i. Start IDLE

ii. Navigate to the File menu and click New Window

iii. Type the following:

print (“Hello World!”)

iv. On the file, menu, click on Save. Type the name of myProgram1.py

v. Navigate to Run and click Run Module to run the program.

The first program that we have written is known as the “Hello World!” and is used to not only provide an introduction to a new computer coding language but also test the basic configuration of the IDE. The output of the program is “Hello World!” Here is what has happened, the Print() is an inbuilt function, it is

prewritten and preloaded for you, is used to display whatever is contained in the () as long as it is between the double quotes. The computer will display anything written within the double quotes.

Exercise: Now write and run the following python programs:

a. print(“I am now a Python Language Coder!”)

b. print(“This is my second simple program!”)

c. print(“I love the simplicity of Python”)

d. print(“I will display whatever is here in quotes such as owyhen2589gdbnz082”)

Now we need to write a program with numbers, but before writing such a program, we need to learn something about Variables and Types.

Remember python is object-oriented and it is not statically typed which means we do not need to declare variables before using them or specify their type. Let us explain this statement; an object-oriented language simply means that the language supports viewing and manipulating real-life scenarios as groups with subgroups that can be linked and shared mimicking the natural order and interaction of things. Not all programming languages are object-oriented; for instance, Visual C programming language is not object-oriented. In programming, declaring variables means that we explicitly state the nature of the variable. The variable can be declared as an integer, long integer, short integer, floating integer, a string, or as a character including if it is accessible locally or globally. A variable is a storage location that changes values depending on conditions.

For instance, number1 can take any number from 0 to infinity. However, if we specify explicitly that int number1 it then means that the storage location will only accept integers and not fractions for instance, fortunately or unfortunately,

python does not require us to explicitly state the nature of the storage location (declare variables) as that is left to the python language itself to figure out that.

Before tackling types of variables and rules of writing variables, let us run a simple program to understand what variables when coding a python program are.

i. Start IDLE

ii. Navigate to the File menu and click New Window

iii. Type the following:

```
num1=4

num2=5

sum=num1+num2

print(sum)
```

iv. On the file, menu, click on Save. Type the name of myProgram2.py

v. Navigate to Run and click Run Module to run the program.

The expected output of this program should be “9” without the double quotes.

Explanation

At this point, you are eager to understand what has just happened and why the print(sum) does not have double quotes like the first programs we wrote. Here is the explanation.

The first line num1=4 means that variable num1(our shortened way of writing number1, first number) has been assigned 4 before the program runs.

The second line num2=5 means that variable num2(our shortened way of writing number2, second number) has been assigned 5 before the program runs.

The computer interprets these instructions and stores the numbers given

The third line sum=num1+num2 tells the computer that takes whatever num1 has been given and add to whatever num2 has been given. In other terms, sum the values of num1 and num2.

The fourth line print(sum) means that display whatever sum has. If we put double quotes to sum, the computer will display the word sum and not the sum of the two numbers! Remember that cliché that computers are garbage in and garbage out. They follow what you give them!

Note: + is an operator for summing variables and has other users that will be discussed later.

Now let us try out three exercises involving numbers before we explain types of variables and rules of writing variables so that you get more freedom to play with variables. Remember variables values vary for instance num1 can take 3, 8, 1562, 1.

Follow the steps of opening the Python IDE and do the following:

i. The output should be 54

```
num1=43

num2=11

sum=num1+num2

print(sum)
```

ii. The output should be 167

```
num1=101

num2=66

sum=num1+num2

print(sum)
```

iii. The output should be 28

num1=9

num2=19

sum=num1+num2

print(sum)

Variables

We have used num1, num2, and sum and the variable names were not just random, they must follow certain rules and conventions. Rules are what we cannot violate while conventions are much like the recommended way. Let us start with the rules:

The Rules of When Naming Variables in Python

i. Variable names should always start with a letter or an underscore, i.e.

num1

_num1

ii. The remaining part of the variable name may consist of numbers, letters, and underscores, i.e.

number1

num_be_r

iii. Variable names are case sensitive meaning that capital letters and non-capital letters are treated differently.

Num1 will be treated differently with num1.

Exercise

Write/suggest five variables for:

a. Hospital department.

b. Bank.

c. Media House.

Given scri=75, scr4=9, sscr2=13, Scr=18

d. The variable names above are supposed to represent scores of students. Rewrite the variables to satisfy Python variable rules and conventions.

Conventions When Naming Variables in Python

As earlier indicated, conventions are not rules per se are the established traditions that add value and readability to the way we name variables in Python.

i. Uphold readability. Your variables should give a hint of what they are handling because programs are meant to be read by other people other than the person writing them.

number1 is easy to read compared to n1. Similarly, first_name is easy to read compared to firstname or firstName or fn. The implication of all these is that both are valid/acceptable variables in python, but the convention is forcing us to write them in an easy to read form.

ii. Use descriptive names when writing your variables. For instance, number1 as a variable name is descriptive compared to yale or mything. In other words, we can write yale to capture values for number1, but the name does not outrightly hint what we are doing. Remember when writing programs; assume another person will maintain them. The person should be able to quickly figure out what the program is all about before running it.

iii. Due to confusion, avoid using the uppercase ‘O,’ lowercase letter ‘l’ and the uppercase letter ‘I’ because they can be confused with numbers. In other terms, using these letters will not be a violation of writing variables, but their inclusion as variable names will breed confusion.

Exercise 1

Re-write the following variable names to (1) be valid variable names and follow (2) conventions of writing variable names.

a. 23doctor

b. line1

c. Option3

d. Mydesk

e. #cup3

Exercise 2

Write/Suggest variable names that are (1) valid and (2) conventional.

a. You want to sum three numbers.

b. You want to store the names of four students.

c. You want to store the names of five doctors in a hospital.

Summary

Variables are storage locations that a user specifies before writing and running a python program. Variable names are labels of those storage locations. A variable holds a value depending on circumstances. For instance, doctor1 can be Daniel, Brenda or Rita. Patient1 can be Luke, William or Kelly. Variable names are written by adhering to rules and conventions. Rules are a must while conventions are optional but recommended as they help write readable variable names. When writing a program, you should assume that another person will examine or run it without your input and thus should be well written. The next chapter will discuss Variables. In programming, declaring variables means that we explicitly state the nature of the variable. The variable can be declared as an integer, long integer, short integer, floating integer, a string, or as a character

including if it is accessible locally or globally. A variable is a storage location that changes values depending on conditions. Use descriptive names when writing your variables.

Types of Variables

Now that we have defined what are variables are and the rules to write variable names let us explore different types of Variables.

i. Numbers

The Python accommodates two kinds of numbers, namely floating point numbers and integer numbers. Python also supports complex numbers. When you sign a value to a number, then a number object is created. For example:

number3 =9

number4=12

Different Numerical Types Supported in Python

- long for example 681581903L
- int for example 11, 123, -151
- float for example 0.5, 23.1, -54.2
- complex for example 4.12j

Exercise

Identify the type of numerical below:

a. 234, 19, 312
b. 4.56, 2.9, 9.3
c. 76189251468290127624471

Identify the numerical type suitable for the following contexts:

d. Salary amount.
e. Counting the number of students in a class.
f. Getting the census figure in an entire country of China.

ii. Strings

A single or double quote in Python is used to indicate strings. The subsets of strings can be taken by using the slice operator ([:]) and []) with indexes beginning at () in the start of the string and operating their way from -1 at the end. Strings can be joined using the + (plus) sign known as the concatenation operator. The asterisk (*) is used as a repetition operator. Remember counting in programming starts from index zero (the first value)!

Example:

i. Start IDLE

ii. Navigate to the File menu and click New Window

iii. Type the following:

```
str = 'Going Deep!'
print str          # Prints a complete string
print str[0]       # Prints first character of the string
print str[3:6]     # Prints characters starting from fourth to seventh
print str[3:]      # Prints string starting from the fourth character
print str * 3      # Prints string three times
print str + "I love Python" # Prints concatenated string
```

iv. On the file, menu click on Save. Type the name of myProgram4.py

Navigate to Run and click Run Module to run the program

The output of the program above should be:

```
Going Deep!
G
ng De
ng Deep!
Going Deep! Going Deep! Going Deep!
Going Deep! I love Python
```

Note: the # (hash sign) is used to indicate a single line comment. A comment is descriptive information about a particular line(s) of code. The comment is normally ignored by when running the program. The comment should be written after the # sign in python. Comments increase the readability of the program written.

Exercise:

You will key in/type the following program statement:

str = 'I think I am now a Programmer.'

a. Write a program statement that will display the entire string/statement above.

b. Write a program statement to display characters of the string from the second character to the sixth.

c. Write a single program statement that will display the entire string two times. (use *).

d. Write a program statement that will add the following at the end of the statement above, “ of Python.”

Keywords and Identifiers in Python

At this point, you have been wondering why you must use print and str in that manner without the freedom or knowledge of why the stated words have to be written in that manner. The words print and str constitute a special type of words that have to be written that way always. Each programming language has a set of keywords. In most cases, some keywords are found across several programming languages. Keywords are case sensitive in python meaning that we have to type them in their lowercase form always. Keywords cannot be used to name a function (we will explain what it is later), name of a variable.

There are 33 keywords in Python, and all are in lowercase save for None, False, and True. They must always be written as they appear below:

for	class	return	is	and	nonlocal	import	yield	raise
try	def	del	elif	with	pass	lambda	except	
as	from	if	while	in	not	global	else	
finally	None	True	or	False	assert	continue	break	

Note: The print() and str are functions, but they are inbuilt/preloaded functions in Pythons. Functions are a set of rules and methods that act when invoked. For instance, the print function will display output when activated/invoked/called. At this point, you have not encountered all of the keywords, but you will meet them gradually. Take time to skim through, read and try to recall as many as you can.

Exercise

Identify what is wrong with the following variable names (The exercise requires recalling what we have learned so far)

a. for=1
b. yield=3
c. 34ball
d. m

Comments and Statements

Statements in Python

A statement in Python refers to instructions that a Python interpreter can work on/execute. An example is str=’ I am a Programmer’ and number1=3. A statement having an equal sign(=) is known as an assignment statement. They

are other types of statements such as the if, while, and for which will be handled later.

Exercise

a. Write a Python statement that assigns the first number value of 18.

b. Write a programming statement that assigns the second number value of 21.

c. What type of statements are a. and b. above?

Multi-line Python Statement

Spreading a statement over multiple lines is possible. Such a statement is known as a multi-line statement. The termination of a programming statement is denoted by a new line character. To spread a statement overs several lines, in Python, we use the backslash (\) known as the line continuation character. An example of a multi-line statement is:

```
sum=3+6+7+\
    9+1+3+\
    11+4+8
```

The example above is also known as an explicit line continuation. In Python, the square brackets [] denotes line continuation similar to parenthesis/round brackets (), and lastly braces {}. The above example can be rewritten as

```
sum=(3+6+7+
    9+1+3+
    11+4+8)
```

Note: We have dropped the backslash(\) known as the line continuation character when we use the parenthesis(round brackets) because the parenthesis is doing the work that the line continuation \ was doing.

Question: Why do you think multi-line statements are necessary we can simply write a single line, and the program statement will run just fine?

Answer: Multi-line statements can help improve formatting/readability of the entire program. Remember, when writing a program always assume that it is other people who will use and maintain it without your input.

Exercise:

Rewrite the following program statements using multi-line operators such as the \, [],() or {} to improve readability of the program statements.

a. total=2+9+3+6+8+2+5+1+14+5+21+26+4+7+13+31+24
b. count=13+1+56+3+7+9+5+12+54+4+7+45+71+4+8+5

Semicolons are also used when creating multiple statements in a single line. Assume we have to assign and display the age of four employees in a python program. The program could be written as:
employee1=25; employee2=45; employee3=32; employee4=43.

Indentation in Python

Indentation is used for categorization program lines into a block in Python. The amount of indentation to use in Python depends entirely on the programmer. However, it is important to ensure consistency. By convention, four whitespaces are used for indentation instead of using tabs. For example:

```
for r in range(2,12):
   print(r)
   if r == 6:
      break
```

Note: We will explain what kind of program of this is later.
Indentation in Python also helps make the program look neat and clean. Indentation creates consistency. However, when performing line continuation indentation can be ignored. Incorrect indentation will create an indentation error.

Correct python programs without indentation will still run, but they might be neat and consistent from human readability view.

Comments in Pythons

When writing python programs and indeed any programming language, comments are very important. Comments are used to describe what is happening within a program. It becomes easier for another person taking a look at a program to have an idea of what the program does by reading the comments in it. Comments are also useful to a programmer as one can forget the critical details of a program written. The hash (#) symbol is used before writing a comment in Python. The comment extends up to the newline character. The python interpreter normally ignores comments. Comments are meant for programmers to understand the program better.

Example

i. Start IDLE

ii. Navigate to the File menu and click New Window

iii. Type the following:

```
#This is my first comment
#The program will print Hello World
Print('Hello World')   #This is an inbuilt function to display
```

iv. On the file, menu, click on Save. Type the name of myProgram5.py

Navigate to Run and click Run Module to run the program

Exercise.

This exercise integrates most of what we have covered so far.

a. Write a program to sum two numbers 45, and 12 and include single line comments at each line of code.

b. Write a program to show the names of two employees where the first

employee is “Daisy” and the second employee is “Richard.” Include single comments at each line of code.

c. Write a program to display the student registration numbers where the student names and their registration are: Yvonne=235, Ian=782, James=1235, Juliet=568.

Multi-line Comments

Just like multi-line program statements we also have multi-line comments. There are several ways of writing multi-line comments. The first approach is to type the hash (#) at each comment line starting point.

For Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
#I am going to write a long comment line
#the comment will spill over to this line
#and finally end here.
```

The second way of writing multi-line comments involves using triple single or double quotes: ‘’’ or”. For multi-line strings and multi-line comments in Python, we use the triple quotes. Caution: When used in docstrings they will generate extra code, but we do not have to worry about this at this instance.

Example:

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

“This is also a great
illustration of
a multi-line comment in Python.”

Python’s Docstring

In Python, docstring refers to words offering description and are written as the initial program statement in a function, module, method, or class definition. (We will handle this later on). Docstrings in Python are written using triple quotes.

Integrated Exercise

This exercise will utilize several concepts that we covered earlier.

a. Given the following program statement: Color1=’red’; color1=’blue’; CoLor1=’yellow’ explain why all the three will be treated as different variables in Python.

b. Consider the following Python program and identify what is wrong with it.

```
student1_age=23                          #This is the age of the first student
student2_age=19                          #This is the age of the student
sotal_age=student1_age +student2_age     #Getting the sum of the ages of the
print(age)                               #Displaying their ages
```

c. Rewrite b. above to be a valid Python program

Operators (Basic) in Python

So far we have been using the summation (+) operator, and it also doubles up as a concatenation operator (appending statements). However, we want to expand our list of operators, and this leads us to basic operators in Python.

Arithmetic Operators

The multiplication (*), division (/), subtraction (-), and addition (+) are the

arithmetic operators used to manipulate numbers.

Exercise

Write the following programs and run it

a. Difference

```
number1=35              #declaring first number
number2= 12             #declaring second number
difference=number2-number1    #declaring what the difference does
print(difference)          #Calling the print function to display what difference has
```

b. Multiplication

```
number1=2              #declaring first number
number2= 15            #declaring second number
product=number1*number2       #declaring what the product does
print(product)             #Calling the print function to display what product has
```

c. Division

```
number1=10             #declaring first number
number2= 50            #declaring second number
division=number2/number1       #declaring what the division does
print(division)            #Calling the print function to display what product has
```

Modulus

The modulus operator is used to return the integer remainder after division. The modulus=dividend%divisor.

Example

Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
number1=2               #declaring first number
number2= 15               #declaring second number
remainder=number2%number1    #declaring what the remainder does
print(remainder)              #Calling the print function to display remainder has
```

Squaring and Cubing in Python

Squaring a number-number**2
Cubing a number-number**3

Example:
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

Square of 3 in Python will be 3**2
Cube of 5 in Python will be 5**3

a. Square of 3

Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
number=3                #declaring  variable number and assigning value 3
square=number**2
```

```
print(square)              #Calling the print function to display what square has
```

b. Cube of 5

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=5              #declaring  variable number and assigning value 5
cube=number**3
print(cube)           #Calling the print function to display what cube has
```

Exercise

Use python operators to write and run a python program that finds the following:

a. Cube of 7
b. Square of 15
c. Cube of 6
d. Square of 11
e. Cube of 8
f. Square of 13

Note: We can still multiply 2 two times to get the square of 2. The reason for using the square and cube operators is to help us write compact and efficient code. Remember that the interpreter goes through each line including comments only that it ignores comments. Using the cube and square operators helps compact code and increase the efficiency of interpretation including troubleshooting as well as human readability of the code.

Operators with String in Python

In Python certain operators are used to help in concatenating strings. The addition sign is used to concatenate strings in Python.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
status=” I am happy I know” + “how to write programs in Python.”
print(status)
Python Multiplication of a string to create a sequence
many_words=”Great Programmer” * 5
print(many_words)
```

Exercise

a. Use a concatenation operator to join the following strings in Python

I have realized
that programming is a passion,
dedication and frequent practice.

b. Use an operator to generate ten times the following string

Happy

Summary

We have covered what constitutes variables in Python. Variables are named storage locations, and for this reason, we have variable names. There are

numerical and string variables. These are known as variable types. In handling variables and indeed any other aspect of Python we will encounter and use special words known as reserved words or keywords. Keywords are fixed and must be typed the way specified. Keywords cannot be used as variable names or identifiers. Comments (preceded using #,’’’ or “) are for human readability aspect of a Python program. Indentation is used in Python to group lines of codes into blocks. Different types of operators such as the arithmetic operators and string operators are used to allow for manipulation of variables supported by the user and inbuilt functions in Python.

Type Conversion and Type Casting in Python

Type Conversion refers to the process of changing the value of one programming data type to another programming data type. Think of dividing two integers that lead to decimal numbers. In this case, it is necessary to convert force the conversion of an integer into a float number. Python has two types of conversion: implicit type conversion and explicit conversion.

i. Implicit Conversion Type

In this case, Python automatically changes one data type to another data type, and the process does not require user involvement. Implicit type conversion is mainly used with the intent of avoiding data loss.

Example

Converting Integer to Float

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=451
number_flo=4.51
number_new=number_int+number_flo
```

```
print(“the type of data of number_int-“, type(number_int))
print(“the type of data of number_flo-“, type(number_flo))
print(“value of number_new-“number_new)
print(“type of data of number_new-“, type(number_new))
```

Explanation

The programming is adding two variables one of data type integer and the other float and storing the value in a new variable of data type float. Python automatically converts small data types into larger data types to avoid prevent data loss. This is known as implicit type conversion.

Important

Python cannot, however, convert numerical data types into string data type or string data type into numerical data type implicitly. Attempting such a conversion will generate an error.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=432              #lower data type
number_str=”241”        #higher data type
print(“The type of data of number_int-“, type(number_int))
print(“The type of data of number_str-“, type(number_str))
print(number_int+number_str)
```

Challenge: Can you guess why this is occurring? Python interpreter is unable to understand whether we want concatenation or addition precisely. When it assumes the concatenation, it fails to locate the other string. When it assumes the addition approach, it fails to locate the other number. The solution to the error above is to have an explicit type conversion.

ii. Explicit Conversion

Here, programmers convert the data type of a named programming object to the needed data type. Explicit conversion is attained through the functions float(), int(), and str() among others.

The syntax for the specific or explicit conversion is:

(needed_datatype) (expression)

Illustration

Summing of a string and integer using by using explicit conversion

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_int=431
number_str=”231”
print(“Type of data of number_int-“, type(number_int))
print(“Type of data number_str prior to Type Casting-“, type(number_str))
number_str=int(number_str)
number_sum=number_int+number_str
print(“Addition of number_int and number_str-“, number_sum)
print(“Type of data of the sum-“, type(number_sum))
```

Note: Running this program will display the data types and sum and display an integer and string.

Explanation

In the above program, we added number_str and number_int variable. We then converted number_str from string(higher data type) to integer(lower data type)type using int() function to perform the summation. Python manages to add the two variables after converting number_str to an integer value. The

number_sum value and data type are an integer data type.

Recap:
The conversion of the object from one data type to another data type is known as type conversion. The python interpreter automatically performs implicit type conversion. Implicit type conversion intends to enable Python to avoid data loss. Typecasting or explicit conversion happens when the data types of object are converted using the predefined function by the programmer. Unlike implicit conversion, typecasting can lead to data loss as we enforce the object to a particular data type.

The previous explicit conversion/typecasting can be written as:
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
number_int=431              #int data type
number_str="231"            #string data type
number_str=int(number_str)          #int function converting string into int data type
number_sum=number_int+number_str    #variable number_sum
print("Addition of number_int and number_str-", number_sum)
```

Note: Explicit conversion/Type casting requires some practice to master, but it is easy.

The trick is here:
the variable name to convert=predefined function for the desired data type (variable name to convert)

Exercise:

Use the knowledge of type casting/explicit data conversion to write a program to compute the sum of:

a. score_int=76
score_str=”61.”

b. count_str=231
count_str=”24”

Use the knowledge of type casting/explicit data conversion to write a program find the product of:

c. number_int=12
number_str=”5.”

d. odds_int=6
odds_str=”2”

e. minute_int=45
minute_str=”7”

Input, Output, and Import in Python

In Python input and output (I/O) tasks are performed by inbuilt functions. The print() performs output/display, while input() performs input tasks. The print() is used to output data to standard output devices such as a screen. However, it is also possible to copy data to a file. You are now familiar with the print function written as a print(). It is an inbuilt function because we do not have to write it and specify what and how it works. They are normally two main types of functions, inbuilt/built-in and user-defined functions that we will handle later. When we want the print function to reference or pick the value of a variable, we do not use double quotes.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=6    #variable definition and assignment
print(number)    #displaying value stored in variable number
```

Note: Now assume we want to offer some explanation to the user running this program as he or she will only 6 and does not understand why 6 appears on the screen.

```
number=6
print("This is a number stored in the variable number, it is:" number)
```

Note: We can display string and other data types in the print function. A comma is generally used to demarcate the end of one data type and the beginning of another.

Exercise

a. Given score=5, write a Python program that displays "The score is 5". The program should reference/extract the value of score in its print function.

b. Given age=26, write a Python to display "Richard's age is 26". The program should reference/extract the value of score in its print function.

c. Given marks=87, write a Python program that displays "The average marks in the class is 87". The program should reference/extract the value of score in its print function.

Important: new line in Python is gotten by "\n" without the double quotes and outside the string or line code.

Formatting Output

It may become necessary to play around with the layout of the output to make it appealing, formatting. The str.format() method is used to format output and is visible/accessible to any string object.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
lucy=3; brian= 7
print('The age of lucy is {} and brian is {}'.format(lucy,brian))
```

Note: The expected output is "The age of lucy is 3, and Brian is 7".

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
print('Brian loves {1} and {0}'.format('soccer', 'movies'))
#The expected output will be: He likes soccer and moves
print('he likes {1} and {0}'.format('soccer', 'movies'))
#Output: He likes movies and soccer
```

Note: The use of tuple index/ numbers to specify the order of displaying frees up the programmer as one does not have to retype the entire string when switching the order of the variables.

Exercise

Given print(‘She studies {0} and {1}’.format(‘Health’, ‘ICT’))

a. Rewrite the Python program to display “She studies ICT and Health.”
b. Rewrite the Python program to display “She studies Health and Health.”
c. Rewrite the Python program to display “She studies ICT and ICT.”

Challenge: where in real life the concept above (tuple index/specifying order of display) may be useful where you don’t have to rewrite entire lines of codes. For instance, think of a website that asks users to register for an account but begins with asking surname before the first name. Now assume that the ICT team has recommended that the users should be asked their first name first before the surname. Write a simple Python simulation program to demonstrate how the tuple index concept can increase efficiency, readability, and maintainability of code.

Input in Python

So far our Python programs have been static meaning that we could not key in as everything was given to the variable before running the program. In real life, applications allow users to type in values to a program and wait for it to act on the values given. The input function (input()) is used to accept input of values from the user.

Syntax/way of using it in Python

variable name=input(‘option to include a message to the user on what is happening/can leave it out’).

Example

A program that accepts numerals from users

number=input(‘Type your number here:’) #Will display: Type a number here:

Important: Python interpreter at this stage will treat any numbers entered as a string. For Python interpreter to treat the keyed-in number as the number, we must convert the number using type conversion functions for numbers which are

int() and float().

To convert the number into an integer, after input from the user does the following:

int(‘number’)

To convert the number into a float, after input from the user does the following:

float(‘number’)

Exercise

a. Write a Python program to accept numerical data from users for their age, to capture the age of a user

b. Use explicit type conversion to convert the string entered into a floating value in a.

c. Write a Python program to accept salary figure input from users.

d. Use explicit type conversion to convert the string into a floating value in c.

e. Write a program to accept the count of students/number of students’ in a class from users.

f. Use explicit type conversion to convert the string entered into an integer data type in e.

Import in Python

The programs we have run so far are small, but in reality, a program can be hundreds to ten thousand lines of code. In this case, a large program is broken into smaller units called modules. These modules are related but on different files normally ending with the extension .py. Python allows importing of a module to another module using the keyword import. Analogy: You probably have some of your certificates scanned and stored in your Google drive, have your notebook in your desk, have a passport photo in your phone external storage, and a laptop in your room. When writing an application for an internship, you will have to find a way of accessing all these resources, but in

normal circumstances, you will only work with a few even though all of them are connected. The same is true for programs.

Example
Assume we need to access the math pi that is a different module. The following program will illustrate:
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
import.math        #referencing to the contents of math module
print(math.pi)      #Now utilizing the features found in that referenced math
```

Namespace and Scope in Python

The identifier is simply a named object. Python handles every item as an object.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
number=3
print(id(3)=‘, id(3))
print(‘id(number)=’, id(number))
```

Note: Both are referring to the same object.

Namespace in Python

When we start the Python interpreter, a namespace containing all inbuilt names is created as long the workspace remains application is active. Inbuilt functions such as print() and id() exist throughout the program.

Built-in Namespace: These are functions, methods, and associated data that immediately accessible as soon the Python interpreter loads and as such are accessible to each instance and area of the workspace.

Global Namespace: This involves the contents of a module that are accessible throughout the module. Modules can have several functions and methods.
Local Namespace: Mostly for user-defined functions, a local namespace is restricted to the particular function and outside the function access is not implicitly possible.

Variable Scope

Even though there might be several unique namespaces specified, it may not be possible to access all of the namespaces given because of scope. The concept of scope refers to a segment of the program from where we access the namespace directly without any prefix. The following are the scope types:

i. Scope containing local names, current function.
ii. Scope containing global names, the scope of the module.
iii. Scope containing built-in names of the outermost scope.

Summary

Type Conversion refers to the process of changing the value of one data type to another data type. Think of dividing two integers that lead to decimal numbers. In this case, it is necessary to convert force the conversion of an integer into a float number. Python has two types of conversion: implicit type conversion and explicit conversion. In Python input and output (I/O) tasks are performed by inbuilt functions. The print() performs output/display, while input() performs input tasks. Namespace in Python refers to a collection of names. While different namespaces can co-exist at an instance, they are completely isolated. When we start the Python interpreter, a namespace containing all inbuilt names is created as long the workspace remains application is active.

Flow Control in Python

Before tackling flow control, it is important we explore logical operators.
Comparison operators are special operators in Python that evaluate to either True or False state of the condition.
Program flow control refers to a way in which a programmer explicitly species the order of execution of program code lines. Normally, flow control involves placing some condition (s) on the program code lines. In this chapter, we will explore and test various flow controls in Python.

if…else Flow Control

The if..else statement in Python is a decision making when executing the program. The if…else statement will ONLY execute code if the specified condition exists.

The syntax of if…else in Python
if test expression:
Statement(s)

Explanation
The python program will only execute the statements(s) if the test expression is True. The program first evaluates the test expression before executing the statement(s). The program will not execute the statement(s) if the test expression is False. By convention, the body of it is marked by indentation while the first is not indented line signals the end.

Challenge: Think of scenarios, real-life, where the if…else condition is required.

- If you have not enrolled for a course, then you cannot sit for the exam else sit for the exam.
- If you have paid for house rent then you will be issued with acknowledgment

receipt else request for more time.

- If you are a licensed driver then you can drive to school else you hire a taxi.
- If you are tired then you can watch movies else can complete the essay.
- If you are an ethical person then you will acknowledge your mistake else you will overlook the damage caused.
- If you are committed to programming then you will practice daily else you will lose interest.
- If you have signed for email alerts you will be updated frequently else you will have to check the website daily.
- If you plead guilty to all accounts you are likely to be convicted else the merit of your case will depend on cross-examination of witnesses and evidence presented.

Note: When we use the if statement alone without the else part, it will only print/display if the condition is true, it will not cater for the alternative, the case where the first condition is not present.

Example 1

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=5
if number>0          #The comparison operator
   print(number, “The number is a positive number”)
```

Explanation

The program contains the if the condition that tests if the given number satisfies the if condition, “is it greater than 0” since 5 is greater than zero, the condition is satisfied the interpreter is allowed to execute the next statement which is to extract and display the numerical value including the string message. The test condition in this program is “number>0. But think of when the condition is not

met, what happens? Let us look at Example 2.

Example 2
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
number=-9
if number>0:
   print(number, “This is a positive number”)
```

Explanation
The program contains only the if statement which tests the expression by testing of -9 is greater than zero since it is not the interpreter will not execute the subsequent program code lines. In real life, you will want to provide for an alternate in case the first condition is not met. This program will not display anything when executed because the if condition has not been met. The test condition in this program is “number>0.

Exercise: Write programs in Python using if statement only to perform the following:

a. Given number=7, write a program to test and display only even numbers.
b. Given number1=8, number2=13, write a program to only display if the sum is less than 10.
c. Given count_int=57, write a program that tests if the count is more than 45 and displays, the count is above the recommended number.
d. Given marks=34, write a program that tests if the marks are less than 50 and display the message, the score is below average.
e. Given marks=78, write a program that tests if the marks are more than 50 and display the message, great performance.

f. Given number=88, write a program that tests if the number is an odd number and displays the message, Yes it is an odd number.

g. Given number=24, write a program that tests and displays if the number is even.

h. Given number =21, write a program that tests if the number is odd and displays the string, Yes it is an odd number.

Note The execution of statements after the if expression will only happen where the if the expression evaluates to True, otherwise the statements are ignored.

if…else Statement in Python

The if…else syntax

```
if test condition:
    Statements
else:
    Statements
```

The explanation the if statement, the if…else statement will execute the body of if in the case that the tests condition is True. Should the if…else tests expression evaluate to false, the body of the else will be executed? Program blocks are denoted by indentation. The if…else provides more maneuverability when placing conditions on the code.

Example

A program that checks whether a number is positive or negative

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_mine=-56
if(number<0):
    print(number_mine, “The number is negative”)
```

```
else:
  print(number_mine, “The number is a positive number”)
```

Exercise

Write a Python program that uses if..else flow control to perform the following

a. Given number=9, write a program that tests and displays whether the number is even or odd.

b. Given marks=76, write a program that tests and displays whether the marks are above pass mark or not bearing in mind that the pass mark is 50.

c. Given number=78, write a program that tests and displays whether the number is even or odd.

d. Given marks=27, write a program that tests and displays whether the marks are above pass mark or not bearing in mind that the pass mark is 50.

Challenge: Write a program that accepts age input from the user, explicitly coverts the age into integer data types, then uses if…else flow control to tests whether the person is underage or not, the legal age is 21. Include comments and indentation to improve the readability of the program.

if…elif…else Flow Control Statement in Python

Now think of scenarios where we need to evaluate multiple conditions, not just one, not just two but three and more. Think of where you have to choose team member, if not Richard, then Mercy, if not Richard and Mercy then Brian, if not Richard, Mercy, and Brian then Yvonne. Real-life scenarios may involve several choices/conditions that have to be captured when writing a program.

```
if...elif..else Syntax
if test expression:
   Body of if
elif test expression:
   Body of elif
else:
   Body of else
```

Remember that the elif simply refers to else if and is intended to allow for checking of multiple expressions. The if block is evaluated first, then elif block(s), before the else block. In this case, the else block is more of a fallback option when all other conditions return false. Important to remember, despite several blocks available in if..elif..else only one block will be executed.

Example: Three conditions covered but the only one can execute at a given instance.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number_mine=87
if(number>0):
  print(number_mine, "This is a positive number")
elif(number_mie==0):
   print(number_mine, "The number is zero")
else:
   print(number_mine, "The number is a negative number")
```

Explanation: There are three possibilities but at any given instance the only condition will exist and this qualifies the use of if family flow control statement. For three or more conditions to evaluate, the if…elif..else flow statement merits.

Nested if Statements in Python

Sometimes it happens that a condition exists but there are more sub-conditions that need to be covered and this leads to a concept known as nesting. The amount of statements to nests is not limited but you should exercise caution as you will realize nesting can lead to user errors when writing code. Nesting can also complicate maintaining of code. The only indentation can help determine the level of nesting.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
my_charact=str(input("Type a character here either 'a', 'b' or 'c':"))
if (my_charact='a'):
  if(my_charact='a'):
    print("a")
  else if:
    (my_charact='b')
    print("b")
else:
 print("c")
```

Exercise
Write a program that uses the if..else flow control statement to check the non-leap year and display either scenario. Include comments and indentation to enhance the readability of the program.
for Loop in Python.
Indentation is used to separate the body of for loop in Python.
Note: Simple linear list takes the following syntax:
Variable_name=[values separated by a comma]

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
numbers=[12, 3,18,10,7,2,3,6,1]    #Variable name storing the list
sum=0                    #Initialize sum before usage, very important
for cumulative in numbers:        #Iterate over the list
sum=sum+cumulative
print("The sum is" ,sum)
```

Exercise
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

Write a Python program that uses the for loop to sum the following lists.

a. marks=[3, 8,19, 6,18,29,15]
b. ages=[12,17,14,18,11,10,16]
c. mileage=[15,67,89,123,76,83]
d. cups=[7,10,3,5,8,16,13]

range() function in Python

The range function (range()) in Python can help generate numbers. Remember in programming the first item is indexed 0.
Therefore, range(11) will generate numbers from 0 to 10.

Example
Start IDLE.

Navigate to the File menu and click New Window.
Type the following:

print(range(7))
The output will be 0,1,2,3,4,5,6

Exercise:
Without writing and running a Python program what will be the output for:

a. range(16)
b. range(8)
c. range(4)

Using range() and len() and indexing
Solved Exercise
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
food=['omelet', 'fish','rice']
for j in range(len(food)):
  print("I prefer", food[j])
```

The output of this program will be:
I prefer omelet
I prefer fish
I prefer jazz

Exercise
Write a Python program to iterate through the following list and include the message I listen to (each of the music genre). Use the for loop, len() and range(). Refer to the previous example on syntax.

folders=[‘Rumba’, ‘House’, ‘Rock’]

Using for loop with else

It is possible to include a for loop with else but as an option. The else block will be executed if the items contained in the sequence are exhausted.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
marks=[12, 15,17]
for i in marks:
  print(i)
else:
  print(“No items left”)
```

Challenge:
Write a Python program that prints all prime numbers between 1 and 50.

while Loop in Python

In Python, the while loop is used to iterate over a block of program code as long as the test condition stays True. The while loop is used in contexts where the user does not know the loop cycles that are required.

As earlier indicated, the while loop body is determined through indentation.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
number=10      #Assign value
sum=0          #Initialize
j=1
while j<=number:
sum=sum+j
j=j+1      #Update the counter
print("The sum of 0-10 natural numbers is", sum)
```

Caution: Failing to include the value of the counter will lead to an infinite loop.

Exercise

a. Write a Python program that utilizes the while flow control statement to display the sum of all odd numbers from 1 to 10.

b. Write a Python program that employs the while flow control statement to display the sum of all numbers from 11 to 21.

c. Write a Python program that incorporates while flow control statement to display the sum of all even numbers from 1 to 10.

Using While loop with else

If the condition is false and no break occurs, a while loop's else part runs.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
track = 0
while track< 4:
```

```
    print("Within the loop")
track = track + 1
else:
    print("Now within the else segment")
```

Python’s break and continue

Let us use real-life analogy where we have to force a stop on iteration before it evaluates completely. Think of when cracking/breaking passwords using a simple dictionary attack that loops through all possible character combinations, you will want the program immediately it strikes the password searched without having to complete. Again, think of when recovering photos you accidentally deleted using recovery software, you will want the recovery to stop iterating through files immediately it finds items within the specified range. The break and continue statement in Python works in a similar fashion.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
for tracker in "bring":
    if tracker == "i":
        break
    print(tracker)
print("The End")
```

continue statement in Python

When the continue statement is used, the interpreter skips the rest of the code inside a loop for the current iteration only and the loop does not terminate. The loop continues with the next iteration.

The syntax of Python continue
continue

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
for tracker in "bring":
    if tracker == "i":
        continue
    print(tracker)
print("Finished")
```

The output of this program will be:
b
r
n
g
Finished

Analogy: Assume that you are running data recovery software and have specified skip word files (.doc, dox extension). The program will have to continue iterating even after skipping word files.

Exercise
a. Write a Python program using for loop that will break after striking “v” in the string “Oliver”.
b. Write a Python program that will continue after skipping “m” in the string

“Lemon”.

pass Statement in Python

Like a comment, a pass statement does not impact the program as it leads to no operation.

The syntax of pass
pass

Think of a program code that you plan to use in future but is not currently needed. Instead of having to insert that code in future, the code can be written as pass statements.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
my_list={‘k’,’i’,’n’}
for tracker in my_list:
  pass
```

Functions in Python

Functions in Python help split the large code into smaller units. Functions make a program more organized and easy to manage.
In Python functions will assume the syntax form below:

```
def name_of_function (arguments):
   “””docstring”””
statements(s)
```

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
def welcome(salute):
    """The Python function welcomes you to
    the individual passed in as
    parameter"""
    print("Welcome " + salute + ". Lovely Day!")
```

Calling a Function in Python

We can call a function once we have defined it from another function or program.
Calling a function simply involves typing the function name with suitable parameters.
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
welcome('Brenda')
```

The output will be "Welcome Brenda. Lovely Day!'

Exercise
Write a function that when called outputs"Hello (student name), kindly submit your work by Sunday".

Docstring

It is placed after the function header as the first statement and explains in summary what the function does. Docstring should always be placed between

triple quotes to accommodate multiple line strings.
Calling/Invoking the docstring we typed earlier

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

print(welcome._doc_)
The output will be “This function welcomes you to
the individual passed in as
parameter”.
The syntax for calling/invoking the docstring is:
print(function_name. _doc_)

Python function return statement

Return syntax
return [list of expressions]
Explanation
The return statement can return a value or a None object.
Example
Print(welcome(“Richard”)) #Passing arguments and calling the function
Welcome Richard. Lovely Day!
None #the returned value

Example of Returning a Value other None
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
def my_function (balling_option):
   """The function returns the absolute value of the keyed number"""
  if balling_option>=0
     return balling_option
else:
   return - balling_option
The output of the function will be:
print(my_function(3))
print(my_function(-2))
```

Variable Scope and Lifetime in Python Functions

Variables and parameters defined within a Python function have local scope implying they are not visible from outside. In Python, the variable lifetime is valid as long the function executes and is the period throughout that a variable exists in memory. Returning the function destroys the function variables.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def function_my()
   marks=15
print("The value inside the function is:", marks)
marks=37
function_my()
print"The value outside the function is:",marks)
```

Function Types

They are broadly grouped into user-defined and built-in functions. The built-in functions are part of the Python interpreter while the user-defined functions are

specified by the user.

Exercise

Give three examples of built-in functions in Pythons

Function Argument

Calling a function requires passing the correct number of parameters otherwise the interpreter will generate an error.

Illustration

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def salute(name,message):
   """This function welcomes to
   the student with the provided message"""
   print("Welcome",salute + ', ' + message)
welcome("Brenda","Lovely Day!")
```

Note: The function welcome() has two parameters. We will not get any error as has been fed with two arguments. Let us try calling the function with one argument and see what happens:

```
welcome(“Brenda”)    #only one argument passed
```

Running this program will generate an error saying “TypeError: welcome() missing 1 required positional argument. The same will happen when we pass no arguments to the function.

Example 2

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

welcome()
The interpreter will generate an error “TypeError: welcome() missing 2 required positional arguments”.

Keywords Arguments in Python

Python provides a way of calling functions using keyword arguments. When calling functions using keyword arguments, the order of arguments can be changed. The values of a function are matched to the argument position-wise.

Note:
In the previous example function welcome when invoked as welcome(“Brenda”, “Lovely Day!”). The value “Brenda” is assigned to the argument name and “Lovely Day!” to msg.
Calling the function using keywords
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

welcome(name=”Brenda”, msg=”Lovely Day!”)
Keywords not following the order
welcome(msg=”Lovely Day!”, name=”Brenda”)

Arbitrary Arguments

It may happen that we do not have knowledge of all arguments needed to be passed into a function. Analogy: Assume that you are writing a program to welcome all new students this semester. In this case, you do not how many will report.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
def welcome(*names):
“””This welcome function salutes all students in the names tuple.”””
    for name in names:
        print(“Welcome”.name)
welcome("Lucy","Richard","Fridah","James")
```

The output of the program will be:
Welcome Lucy
Welcome Richard
Welcome Fridah
Welcome James

Recursion in Python

The definition of something in terms of itself is called recursion. A recursive function calls other functions.

Example
Python program to compute integer factorials
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
def compute_fact(number):
   """This is a recursive function
To compute the factorial of an integer"""
   if number == 1:
      return 1
   else:
      return (number * compute_fact(number-1))
my_number = 5
print("The factorial of", my_number, "is", comp_fact(my_number))
```

Exercise

Write a Python program to find the factorial of 7.

Python Anonymous Function

Some functions may be specified devoid of a name and these are called anonymous functions. The lambda keyword is used to denote an anonymous function. Anonymous functions are also referred to as lambda functions in Python.

Syntax

lambda arguments: expression.

Lambda functions must always have one expression but can have several arguments.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
double = lambda y: y * 2
# Output: 10
print(double(5))
```

Example 2

We can use inbuilt functions such as filter () and lambda to show only even numbers in a list/tuple.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
first_marks = [3, 7, 14, 16, 18, 21, 13, 32]
fresh_marks = list(filter(lambda n: (n%2 == 0) , first_marks))
# Output: [14, 16, 18, 32]
print(fresh_marks)
```

Lambda function and map() can be used to double individual list items.

Example 3

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
first_score = [3, 7, 14, 16, 18, 21, 13, 32]
fresh_score = list(map(lambda m: m * 2 , first_score))
# Output: [6, 14, 28, 32, 36, 42, 26, 64]
print(fresh_score)
```

Python’s Global, Local and Nonlocal

Python’s Global Variables

Variables declared outside of a function in Python are known as global variables. They are declared in global scope. A global variable can be accessed outside or

inside of the function.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
y= "global"
def foo():
    print("y inside the function :", y)
foo()
print("y outside the function:", y)
```

Explanation
In the illustration above, y is a global variable and is defined as a foo() to print the global variable y. When we call the foo() it will print the value of y.
Local Variables
A local variable is declared within the body of the function or in the local scope.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
def foo():
    x = "local"
foo()
print(x)
```

Explanation

Running this program will generate an error indicating ‘x’ is undefined. The error is occurring because we are trying to access local variable x in a global scope whereas foo() functions only in the local scope.
Creating a Local Variable in Python
Example

A local variable is created by declaring a variable within the function.
def foo():
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
    x = "local"
    print(x)
foo()
```

Explanation
When we execute the code, the output will be:
Local

Python’s Global and Local Variable

Using both local and global variables in the same code.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
y = "global"
def foo():
```

```
    global y
    x = "local"
    y = y * 2
    print(y)
    print(x)
foo()
```

Explanation
The output of the program will be:
global global
local

Explanation
We declared y as a global variable and x as a local variable in the foo(). The * operator issued to modify the global variable y and finally, we printed both y and x.

Local and Global Variables with the same name
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
y=6
def foo():
y=11
   print(“Local variable y-“, y)
foo()
print("Global variable y-", y)
```

Python’s Nonlocal Variables

A Python’s nonlocal variable is used in a nested function whose local scope is unspecified. It is neither global nor local scope.

Example

Creating a nonlocal variable.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
def outer():
    y = "local variable"
    def inner():
        nonlocal y
        y = "nonlocal variable"
        print("inner:", y)
    inner()
    print("outer scope:", y)
outer()
```

Global Keyword in Python

The global keyword I Python allows modification of the variable outside the current scope. The global keyword makes changes to the variable in a local context. There are rules when creating a global keyword:

A global keyword is local by default when we create a variable within a function.

It is global by default when we define a variable outside of a function and you do not need to use the global keyword.

The global keyword is used to read and write a global variable within a function.

The use of global keyword outside a function will have no effect.

Example.

Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
number = 3        #A global variable
def add():
   print(number)
add()
```

The output of this program will be 3.
Modifying global variable from inside the function.

```
number=3                  #a global variable
def add():
   number= number + 4     # add 4 to 3
   print(number)
add()
```

Explanation
When the program is executed it will generate an error indicating that the local variable number is referenced before assignment. The reason for encountering the error is because we can only access the global variable but are unable to modify it from inside the function. Using a global keyword would solve this.

Example.
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

Modifying global variable within a function using the global keyword.

```
number = 3            # a global variable
def add():
```

```
    global number
    number= number + 1 # increment by 1
    print("Inside the function add():", number)
add()
print("In main area:", number)
```

Explanation
When the program is run, the output will be:
Inside the function add(): 4
In the main area: 4
We defined a number as a global keyword within the function add(). The variable was then incremented by 1, variable number. Then we called the add () function to print global variable c.

Creating Global Variables across Python Modules

We can create a single module config.py that will contain all global variables and share the information across several modules within the same program.

Example.
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

Create config.py

```
x=0
y=”empty”
```

Then create an update.py file to modify global variables

```
Import config
config.x=11
config.y=”Today”
```

Then create a main.py file to evaluate the changes in value

```
import config
import update
print(config.x)
print(config.y)
```

Explanation

Running the main.py file will generate:

11

Today

Python Modules

Modules consist of definitions as well as program statements.

An illustration is a file name config.py which is considered as a module. The module name would be config. Modules are sued to help break large programs into smaller manageable and organized files as well as promoting reusability of code.

Example: Creating the First module

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
Def add(x, y):
“””This is a program to add two
   numbers and return the outcome"""
outcome=x+y
return outcome
```

Module Import

The keyword import is used to import.

Example
Import first
The dot operator can help us access a function as long as we know the name of the module.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

first.add(6,8)
Explanation

import statement in Python

The import statement can be used to access the definitions within a module via the dot operator.
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

import math
print("The PI value is", math.pi)
Import with renaming

Example
Start IDLE.
Navigate to the File menu and click New Window.

Type the following:

```
import math as h
  print(“The PI value is-“,h.pi)
```

Explanation
In this case, h is our renamed math module with a view helping save typing time in some instances. When we rename the new name becomes valid and recognized one and not the original one.
From…import statement Python.

It is possible to import particular names from a module rather than importing the entire module.
Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
from math import pi
print("The PI value is-", pi)
```

Importing all names

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
from math import*
print("The PI value is-", pi)
```

Explanation

In this context, we are importing all definitions from a particular module but it is encouraged norm as it can lead to unseen duplicates.

Module Search Path in Python

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
import sys
sys.path
```

Python searches everywhere including the sys file.

Reloading a Module

Python will only import a module once increasing efficiency in execution.

```
print(“This program was executed”)
import mine
```

Reloading Code
Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
import mine
import mine
import mine
mine.reload(mine)
```

Dir() built-in Python function

For discovering names contained in a module, we use the dir() inbuilt function.

Syntax
dir(module_name)

Python Package

Files in python hold modules and directories are stored in packages. A single package in Python holds similar modules. Therefore, different modules should be placed in different Python packages.

Data types in Python

Numbers

As mentioned earlier, Python accommodates floating, integer and complex numbers. The presence or absence of a decimal point separates integers and floating points. For instance, 4 is integer while 4.0 is a floating point number.

On the other hand, complex numbers in Python are denoted as r+tj where j represents the real part and t is the virtual part. In this context, the function type() is used to determine the variable class. The Python function instance() is invoked to make a determination of which specific class function originates from.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
number=6
  print(type(number))              #should output class int
print(type(6.0))              #should output class float
complex_num=7+5j
```

```
print(complex_num+5)
print(isinstance(complex_num, complex)) #should output True
```

Important: Integers in Python can be of infinite length. Floating numbers in Python are assumed precise up to fifteen decimal places.

Number Conversion

This segment assumes you have prior basic knowledge of how to manually or using a calculator to convert decimal into binary, octal and hexadecimal. Check out the Windows Calculator in Windows 10, Calculator version Version 10.1804.911.1000 and choose programmer mode to automatically convert.

Programmers often need to convert decimal numbers into octal, hexadecimal and binary forms. A prefix in Python allows denotation of these numbers to their corresponding type.

Number System	Prefix
Octal	‘0O’ or '0o'
Binary	‘0B' or '0b'
Hexadecimal	'0X or '0x'

Example

```
print(0b1010101)          #Output:85
print(0x7B+0b0101)        #Output: 128 (123+5)
print(0o710)          #Output:710
```

Exercise

Write a Python program to display the following:

a. 0011 11112
b. 7478
c. 9316

Type Conversion

Sometimes referred to as coercion, type conversion allows us to change one type

of number into another. The preloaded functions such as float(), int() and complex() enable implicit and explicit type conversions. The same functions can be used to change from strings.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
int(5.3)          #Gives 5
int(5.9)          #Gives 5
```

The int() will produce a truncation effect when applied to floating numbers. It will simply drop the decimal point part without rounding off. For the float() let us take a look:
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
float(6)          #Gives 6.0
ccomplex('4+2j')      #Gives (4+2j)
```

Exercise
Apply the int() conversion to the following:

a. 4.1
b. 4.7
c. 13.3
d. 13.9

Apply the float() conversion to the following:

e. 7

f. 16

g. 19

Decimal in Python

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
(1.2+2.1)==3.3          #Will return False, why?
```

Explanation

The computer works with finite numbers and fractions cannot be stored in their raw form as they will create an infinite long binary sequence.

Fractions in Python

The fractions module in Python allows operations on fractional numbers.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import fractions
print(fractions.my_fraction(2.5))          #Output 5/2
print(fractions.my_fraction(4))             #Output 5
print(fractions.my_fraction(2,5))          #output 2/5
```

Important

Creating my_fraction from float can lead to unusual results due to the misleading representation of binary floating point.

Mathematics in Python

To carry out mathematical functions, Python offers modules like random and math.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import math
print(math.pi)          #output:3.14159….
print(math.cos(math.pi))  #the output will be -1.0
print(math.exp(10))          #the output will be 22026.4….
print(math.log10(100))      #the output will be 2
print(math.factorial(5))  #the output will be 120
```

Exercise

Write a python program that uses math functions from the math module to perform the following:

a. Square of 34
b. Log1010000
c. Cos 45 x sin 90
d. Exponent of 20

Random function in Python

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
import math
print(random.shuffle_num(11, 21))
y=[‘f’,’g’,’h’,’m’]
```

print(random.pick(y))
random.anypic(y)
print(y)
print(your_pick.random())

Lists in Python

We create a list in Python by placing items called elements inside square brackets separated by commas. The items in a list can be of mixed data types.
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
list_mine=[]              #empty list
list_mine=[2,5,8]             #list of integers
list_mine=[5,"Happy", 5.2]      #list having mixed data types
```

Exercise
Write a program that captures the following in a list: "Best", 26,89,3.9

Nested Lists

A nested list is a list as an item in another list.
Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
list_mine=["carrot", [9, 3, 6], ['g']]
```

Exercise
Write a nested for the following elements: [36,2,1],"Writer",'t',[3.0, 2.5]

Accessing Elements from a List

In programming and in Python specifically, the first time is always indexed zero. For a list of five items, we will access them from index0 to index4. Failure to access the items in a list in this manner will create index error. The index is always an integer as using other number types will create a type error. For nested lists, they are accessed via nested indexing.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
list_mine=[‘b’,’e’,’s’,’t’]
print(list_mine[0])          #the output will be b

print(list_mine[2])          #the output will be s
print(list_mine[3])          #the output will be t
```

Exercise

Given the following list:

your_collection=[‘t’,’k’,’v’,’w’,’z’,’n’,’f’]

a. Write a Python program to display the second item in the list
b. Write a Python program to display the sixth item in the last
c. Write a Python program to display the last item in the list.

Nested List Indexing

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
nested_list=[“Best’,[4,7,2,9]]
print(nested_list[0][1]
```

Python Negative Indexing

For its sequences, Python allows negative indexing. The last item on the list is index-1, index -2 is the second last item and so on.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
list_mine=[‘c’,’h’,’a’,’n’,’g’,’e’,’s’]
print(list_mine[-1])                #Output is s

print(list_mine [-4])       ##Output is n
```

Slicing Lists in Python

Slicing operator(full colon) is used to access a range of elements in a list.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
list_mine=[‘c’,’h’,’a’,’n’,’g’,’e’,’s’]
print(list_mine[3:5])               #Picking elements from the 4 to the sixth
```

Example

Picking elements from start to the fifth

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
print(list_mine[:-6])
```

Example
Picking the third element to the last.

```
print(list_mine[2:])
```

Exercise
Given class_names=[‘John’, ‘Kelly’, ‘Yvonne’, ‘Una’,’Lovy’,’Pius’, ‘Tracy’]

a. Write a python program using slice operator to display from the second students and the rest.
b. Write a python program using slice operator to display the first student to the third using negative indexing feature.
c. Write a python program using slice operator to display the fourth and fifth students only.

Manipulating Elements in a List using the assignment operator.

Items in a list can be changed meaning lists are mutable.
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
list_yours=[4,8,5,2,1]
list_yours[1]=6
print(list_yours)            #The output will be [4,6,5,2,1]
```

Changing a range of items in a list

Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
list_yours[0:3]=[12,11,10]       #Will change first item to fourth item in the list
print(list_yours)                #Output will be: [12,11,10,1]
```

Appending/Extending items in the List

The append() method allows extending the items in the list. The extend() can also be used.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
list_yours=[4, 6, 5]
list_yours.append(3)
print(list_yours)                #The output will be [4,6,5, 3]
```

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
list_yours=[4,6,5]
list_yours.extend([13,7,9])
print(list_yours)                #The output will be [4,6,5,13,7,9]
```

The plus operator(+) can also be used to combine two lists. The * operator can be used to iterate a list a given number of times.

Example
Start IDLE.

Navigate to the File menu and click New Window.
Type the following:

```
list_yours=[4,6,5]
print(list_yours+[13,7,9])              # Output:[4, 6, 5,13,7,9]
print([‘happy’]*4)          #Output:[“happy”,”happy”, “happy”,”happy”]
```

Removing or Deleting Items from a List

The keyword del is used to delete elements or the entire list in Python.
Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
list_mine=[‘t’,’r’,’o’,’g’,’r’,’a’,’m’]
del list_mine[1]
print(list_mine)              #t, o, g, r, a, m
```

Deleting Multiple Elements

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
del list_mine[0:3]
```

Example

```
print(list_mine)              #a, m
```

Delete Entire List

Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
delete list_mine
print(list_mine)              #will generate an error of lost not found
```

The remove() method or pop() method can be used to remove the specified item. The pop() method will remove and return the last item if the index is not given and helps implement lists as stacks. The clear() method is used to empty a list.
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
list_mine=['t','k','b','d','w','q','v']
list_mine.remove('t')
print(list_mine)              #output will be ['t','k','b','d','w','q','v']
print(list_mine.pop(1))           #output will be 'k'
print(list_mine.pop())            #output will be 'v'
```

Exercise
Given list_yours=['K','N','O','C','K','E','D']

a. Pop the third item in the list, save the program as list1.
b. Remove the fourth item using remove() method and save the program as list2
c. Delete the second item in the list and save the program as list3.
d. Pop the list without specifying an index and save the program as list4.

Using Empty List to Delete an entire or specific elements

Start IDLE.

Navigate to the File menu and click New Window.
Type the following:

list_mine=[‘t’,’k’,’b’,’d’,’w’,’q’,’v’]
list_mine=[1:2]=[]
print(list_mine) #Output will be [‘t’,’w’,’q’,’v’]

Summary of List Methods in Python

Method	Description		Method	Description
insert()	- Insert an item at the defined index		copy()	- Returns a shallow copy of the list
append()	- Add an element to the end of the list		count()	- Returns the count of number of items passed as an argument
pop()	- Removes and returns an element at the given index		clear()	- Removes all items from the list
index()	- Returns the index of the first matched item		sort()	- Sort items in a list in ascending order
remove()	- Removes an item from the list		extend()	- Add all elements of a list to the another list
			reverse()	- Reverse the order of items in the list

Exercise

a. Use list access methods to display the following items in reversed order list_yours=[4,9,2,1,6,7]
b. Use the list access method to count the elements in a.
c. Use the list access method to sort the items in a. in an ascending order/default.

Inbuilt Python Functions to Manipulate Python Lists

Method	Description		Method	Description
enumerate()	Return an enumerate object and contains the index and value of all the items of list as a tuple.		len()	Return the length in the list.
sorted()	Return a new sorted list but does not sort the list itself		any()	Return True if any element of the list is true. If the list is empty, return False.
sum()	Return the sum of all elements in the list.		min()	Return the smallest item in the list
max()	Return the largest item in the list.		all()	Return True if all elements of the list are true

Tuple in Python

A tuple is like a list but we cannot change elements in a tuple.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
tuple_mine = (21, 12, 31)
print(tuple_mine)
tuple_mine = (31, "Green", 4.7)
print(tuple_mine)
```

Accessing Python Tuple Elements

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
tuple_mine=['t','r','o','g','r','a','m']
        print(tuple_mine[1])                #output:'r'
```

print(tuple_mine[3]) #output:’g’

Negative Indexing

Just like lists, tuples can also be indexed negatively.

Like lists, -1 refers to the last element on the list and -2 refer to the second last element.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
tuple_mine=[‘t’,’r’,’o’,’g’,’r’,’a’,’m’]
print(tuple_mine [-2])  #the output will be ‘a’
```

Slicing

The slicing operator, the full colon is used to access a range of items in a tuple.

Example

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
tuple_mine=[‘t’,’r’,’o’,’g’,’r’,’a’,’m’]
print(tuple_mine [2:5])  #Output: ‘o’,’g’,’r’,’a’
print(tuple_mine[:-4])  #’g’,’r’,’a’,’m’
```

Important

Tuple elements are immutable meaning they cannot be changed. However, we can combine elements in a tuple using +(concatenation operator). We can also repeat elements in a tuple using the * operator, just like lists.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
print((7, 45, 13) + (17, 25, 76))
print(("Several",) * 4)
```

Important
Since we cannot change elements in a tuple, we cannot delete the elements too. However, removing the full tuple can be attained using the keyword del.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
t_mine=[‘t’,’k’,’q’,’v’,’y’,’c’,’d’]
del t_mine
```

Available Tuple Methods in Python

They are only two methods available for working Python tuples.
count(y)
When called will give the item numbers that are equal to y.
index(y)
When called will give index first item index that is equal to y.

Example
Start IDLE.
Navigate to the File menu and click New Window.

Type the following:

```
t_mine=[‘t’,’k’,’q’,’v’,’y’,’c’,’d’]
print(t_mine.count('t'))
print(t_mine.index('l'))
```

Testing Membership in Tuple

The keyword in us used to check the specified element exists in a tuple.

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
t_mine=[‘t’,’k’,’q’,’v’,’y’,’c’,’d’]
print('a' t_mine)              #Output: True
print('k' in t_mine)                  #Output: False
```

Inbuilt Python Functions with Tuple

Method	Description		Method	Description
enumerate()	Return an enumerate object. It contains the index and value of all the items of tuple as pairs.		tuple()	Convert an iterable to a tuple.
sorted()	Take elements in the tuple and return a new sorted list (does not sort the tuple itself).		max()	Return the largest item in the tuple.
all()	Return True if all elements of the tuple are true (or if the tuple is empty).		sum()	Retrun the sum of all elements in the tuple.
len()	Return the length (the number of items) in the tuple.		min()	Return the smallest item in the tuple
			any()	Return True if any element of the tuple is true. If the tuple is empty, return False.

String in Python.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
string_mine = 'Colorful'
print(string_mine)
string_mine = "Hello"
print(string_mine)
string_mine = "'Hello'"
print(string_mine)
string_mine = """I feel like I have
        been born a programmer"""
print(string_mine)
```

Accessing items in a string

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
str = 'Colorful'
print('str = ', str)
print('str[1] = ', str[1])              #Output the second item
print('str[-2] = ', str[-2])    #Output the second last item
print('str[2:4] = ', str[2:4])         #Output the third through the fifth item
```

Deleting or Changing in Python

In Python, strings are immutable therefore cannot be changed once assigned.

However, deleting the entire string is possible.

Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
del string_mine
```

String Operations

Several operations can be performed on a string making it a widely used data type in Python.
Concatenation using the + operator, repetition using the * operator
Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
string1=’Welcome’
string2=’Again’
        print(‘string1+string2=’,string1+string2)
print(' string1 * 3 =', string1 * 3)
```

Exercise
Given string_a=”I am awake” and string_b=”coding in Python in a pajama”

String Iteration

The ***for control*** statement is used to continually scan through an entire scan until the specified number of times are reached before terminating the scan.
Example
Start IDLE.

Navigate to the File menu and click New Window.
Type the following:

```
tracker=0
for letter in “Welcome Again”:
   if(letter == 'l'):
      tracker += 1
print(tracker,'letters found')
```

Membership Test in String

The keyword in is used to test if a sub string exists.
Example
‘t’ in “triumph’ #Will return True
Inbuilt Python Functions for working with Strings
They include enumerate() and len().The len() function returns the length of the string.

String Formatting in Python

Escape Sequences
Single and Double Quotes
Example
Start IDLE.
Navigate to the File menu and click New Window.
Type the following:

```
print('They said, "We need a new team?"')       # escape with single quotes
# escaping double quotes
print("They said, \" We  need a new team\"")
```

Escape Sequences in Python

The escape sequences enable us to format our output to enhance clarity to the

human user. A program will still run successfully without using escape sequences but the output will be highly confusing to the human user. Writing and displaying output in expected output is part of good programming practices. The following are commonly used escape sequences.

Method	Description		Method	Description
\n	ASCII Linefeed		\b	ASCII Backspace
\"	Double quote		\\	Backslash.
\f	ASCII Formfeed		\a	ASCII Bell
\newline	Backslash and newline ignored		\'	Single quote
\r	ASCII Carriage Return		\t	ASCII Horizontal Tab
\v	ASCII Vertical Tab		\ooo	Character with octal value ooo
\xHH	Character with hexadecimal value HH			

Examples

Start IDLE.

Navigate to the File menu and click New Window.

Type the following:

```
print("D:\\Lessons\\Programming")

print("Prints\n in two lines")
```

Summary

As earlier indicated earlier, integers, floating point, and complex numbers are supported in Python. There are integers, floating and complex classes that help convert different number data types. The presence or absence of a decimal point separates integers and floating points. For instance, 4 is integer while 4.0 is a floating point number. Programmers often need to convert decimal numbers into octal, hexadecimal and binary forms. We can represent binary, hexadecimal and octal systems in Python by simply placing a prefix to the particular number. Sometimes referred to as coercion, type conversion allows us to change one type of number into another.

Inbuilt functions such as int() allow us to convert data types directly. The same functions can be used to convert from strings. We create a list in Python by placing items called elements inside square brackets separated by commas. In programming and in Python specifically, the first time is always indexed zero. For a list of five items, we will access them from index0 to index4. Failure to access the items in a list in this manner will create index error.

Chapter 4: Machine Learning with Python

In this chapter, you will be introduced to ML using Python. To understand ML practically, you will learn how to use the best ML algorithm known as K-Nearest Neighbor using Python. For the rest of this chapter, we shall refer to the algorithm as KNN. You will be implementing KNN on the well-known Iris set of data.

Note: You might want to consider taking up the course on ML with Python or for background on how ML evolved and a lot more consider reading this chapter.

Introduction

Machine Learning evolved from computer science that primarily studies the design of algorithms that can learn from experience. To learn, they need **data** that has certain attributes based on which the algorithms try to find some meaningful predictive patterns. Majorly, ML tasks can be categorized as concept learning, clustering, predictive modeling, etc. The ultimate goal of ML algorithms is to be able to take decisions without any human intervention correctly. Predicting the stocks or weather is a couple of applications of ML algorithms.

There are various ML algorithms like Decision trees, Naive Bayes, Random forest, Support vector machine, K-nearest neighbor, K-means clustering, etc.

From the class of ML algorithms, the one that you will be using today is a k-nearest neighbor. Now, the question is what exactly is the K-Nearest Neighbor algorithm? So let us embark on finding out!

What is a k-Nearest Neighbor?

The KNN or k-nearest neighbor algorithm is a supervised learning algorithm, by

supervising it means that it makes use of the class labels of training data during the learning phase. It is an instance-based ML algorithm, where new data points are classified based on stored, labeled instances (data points). KNN can be used both for classification and regression; however, it is more widely used for classification purposes.

The k in KNN is a crucial variable also known as a hyperparameter that helps in classifying a data point accurately. More, precisely, the k is the number of nearest neighbors you wish to take a vote from when classifying a new data point.

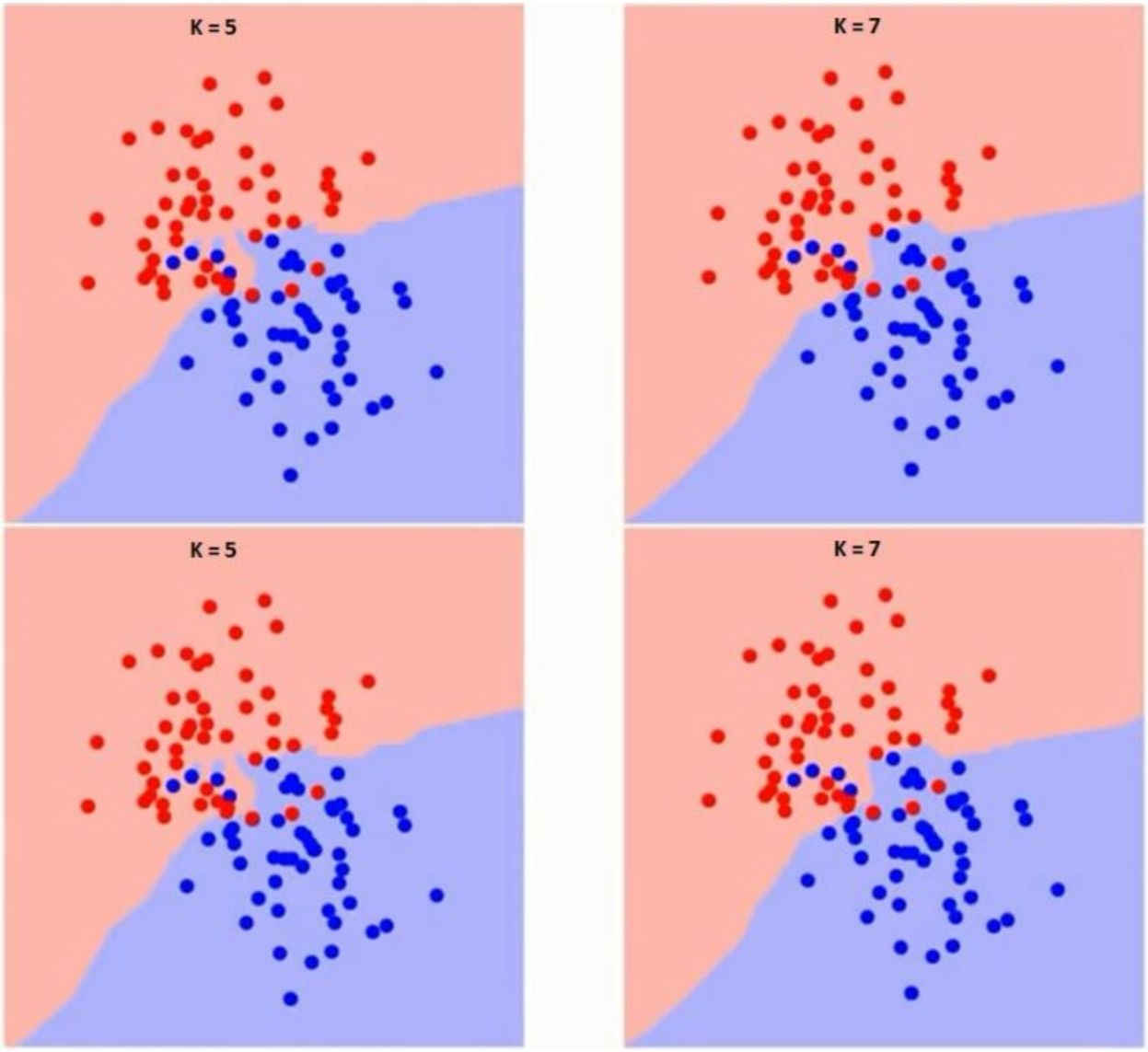


Figure 1. Visualization of KNN Source

You can see as the value of k increases from 1 to 7, the decision boundary between two classes having some data points becomes more Smoother.

Now the question is how does all of this magic happen, wherein every time a new data point comes in, it is classified based on the stored data points?

So, let's quickly understand it in the following ways:

- Firstly, you load all the data and initialize the value of k,
- Then, the distance between the stored data points and a new data point that you want to classify is calculated using various similarity or distance metrics like Manhattan distance (L1), Euclidean distance (L2), Cosine similarity, Bhattacharyya distance, Chebyshev distance, etc.
- Next, the distance values are sorted either in descending or ascending order and top or lower k-nearest neighbors are determined.
- The labels of the k-nearest neighbors are gathered, and a majority vote or a weighted vote is used for classifying the new data point. The new data point is assigned a class label based on a certain data point that has the highest score out of all the stored data points.
- Finally, the predicted class for the new instance is returned.

The prediction can be of two types: either classification in which a class label is assigned to a new data point or regression wherein a value is assigned to the new data point. Unlike classification, in regression, the mean of all the k-nearest neighbors is assigned to the new data point.

Drawbacks of KNN: Firstly, complexity in searching the nearest neighbor for each new data point. Secondly, determining the value of k sometimes becomes a tedious task. Finally, it is also not clear which type of distance metric one should use while computing the nearest neighbors.

Enough of theory right? So, let's load, analyze and understand the data that you will be using in today's small tutorial.

Loading the Iris data

Iris data set consists of 150 samples having three classes namely Iris-Setosa, Iris-Versicolor, and Iris-Virginica. Four features/attributes contribute to uniquely identifying as one of the three classes are sepal-length, sepal-width, petal-length and petal-width.

Feel free to use some other public dataset or your private dataset.

Sklearn is a ML python library that is widely used for data-science related tasks. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means, KNN, etc.. Under sklearn you have a library called datasets in which you have multiple datasets that can be used for different tasks including the Iris dataset, all these datasets can be loaded out of the box. It is pretty intuitive and straightforward. So, let's quickly load the iris dataset.

```
from sklearn.datasets import load_iris
```

load_iris has both the data and the class labels for each sample. Let's quickly extract all of it.

```
data = load_iris().data
```

data variable will be a numpy array of shape (150,4) having 150 samples each having four different attributes. Each class has 50 samples each.

```
data.shape
(150, 4)
```

Let's extract the class labels.

```
labels = load_iris().target
labels.shape
(150,)
```

Next, you have to combine the data and the class labels, and for that, you will use an excellent python library called NumPy.

```
import numpy as np
```

Since data is a 2-d array, you will have to reshape the labels also to a 2-d array.

```
labels = np.reshape(labels,(150,1))
```

Now, you will use the concatenate function available in the numpy library, and you will use axis=-1 which will concatenate based on the second dimension.

```
data = np.concatenate([data,labels],axis=-1)
data.shape
(150, 5)
```

Next, you will import python's data analysis library called pandas which is useful when you want to arrange your data in a tabular fashion and perform some operations and manipulations on the data.
In today's tutorial, you will use pandas quite extensively.

```
import pandas as pd
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'species']
dataset = pd.DataFrame(data,columns=names)
```

Now, you have the dataset data frame that has both data & the class labels that you need!

Before you dive any further, remember that the labels variable has class labels as numeric values, but you will convert the numeric values as the flower names or species.

For doing this, you will select only the class column and replace each of the three numeric values with the corresponding species. You will use inplace=True which will modify the data frame dataset.

```
dataset['species'].replace(0, 'Iris-setosa',inplace=True)
dataset['species'].replace(1, 'Iris-versicolor',inplace=True)
dataset['species'].replace(2, 'Iris-virginica',inplace=True)
```

Let us print the first five rows of the dataset and see what it looks like!

dataset.head(5)

	sepal-length	sepal-width	petal-length	petal-width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Analyze your data

Let's quickly find out how all the three flowers look like when visualized and how different they are from each other not just in numbers but also in real!

Iris Versicolor **Iris Setosa** **Iris Virginica**

(Source)

Let's visualize the data that you loaded above using a scatterplot to find out how much one variable is affected by the other variable or let's say how much correlation is between the two variables.

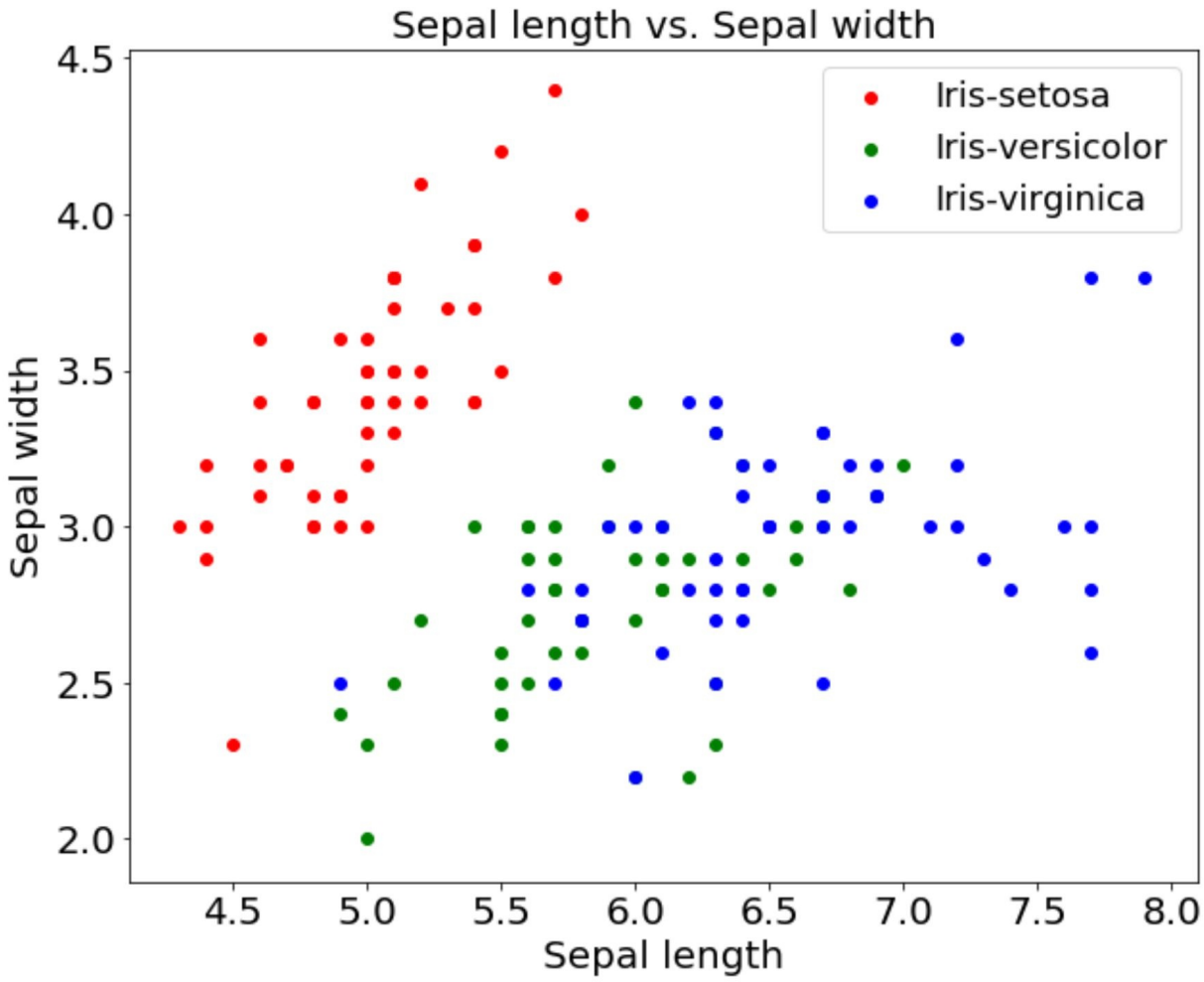
You will use the matplotlib library to visualize the data using a scatterplot.

import matplotlib.pyplot as plt

Note: Are you keen on learning different ways of visualizing the data in python?

Then check out any tutorial about data visualization and also study the code snippet below.

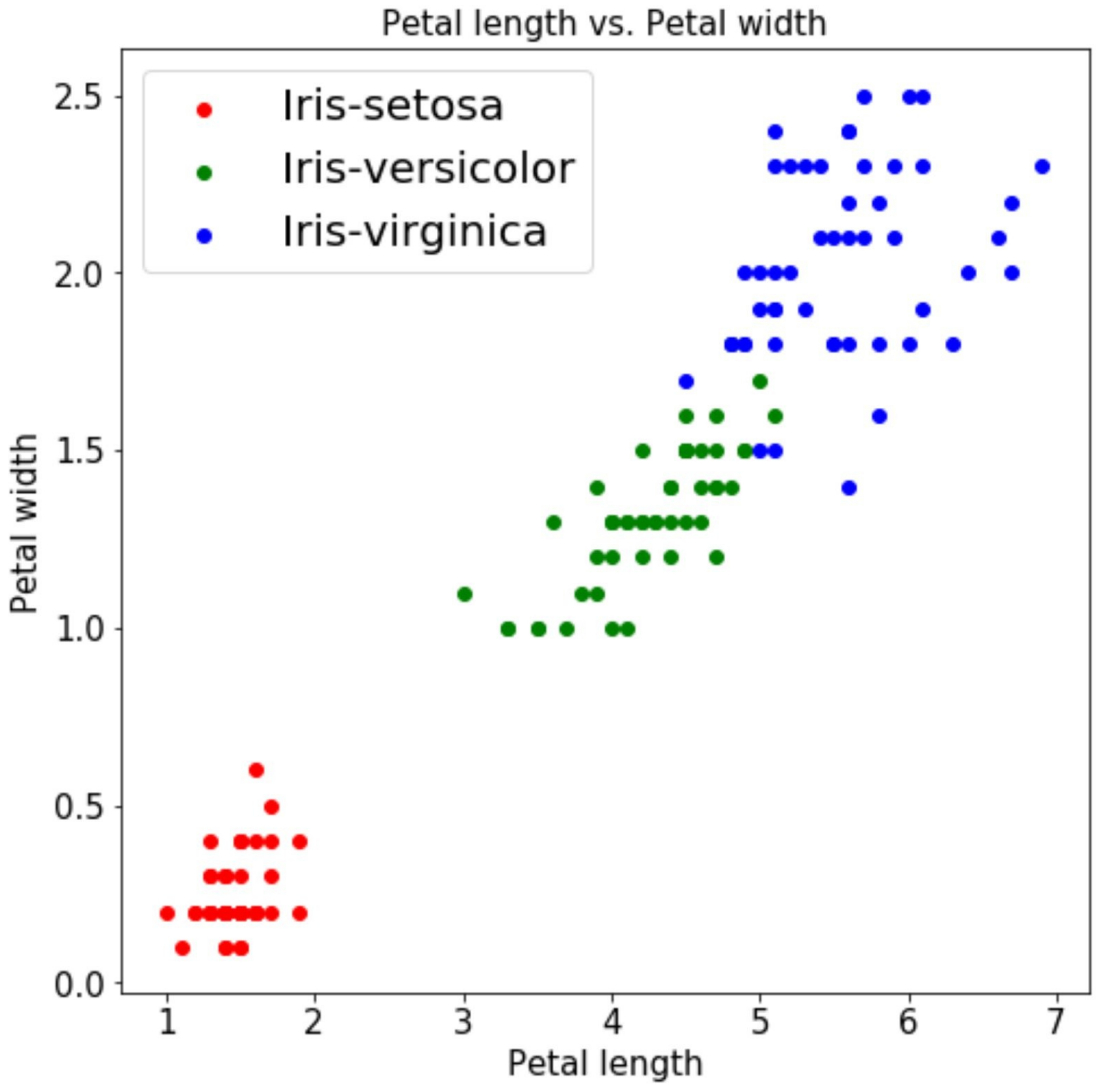
```
plt.figure(4, figsize=(10, 8))

plt.scatter(data[:50, 0], data[:50, 1], c='r', label='Iris-setosa')

plt.scatter(data[50:100, 0], data[50:100, 1], c='g',label='Iris-versicolor')

plt.scatter(data[100:, 0], data[100:, 1], c='b',label='Iris-virginica')

plt.xlabel('Sepal length',fontsize=20)
plt.ylabel('Sepal width',fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.title('Sepal length vs. Sepal width',fontsize=20)
plt.legend(prop={'size': 18})
plt.show()
```



From the above plot, it is very much apparent that there is a high correlation between the Iris setosa flowers w.r.t the sepal length and sepal width. On the other hand, there is less correlation between Iris versicolor and Iris virginica. The data points in versicolor & virginica are more spread out compared to setosa that are dense.

Let's just quickly also plot the graph for petal-length and petal-width.

```
plt.figure(4, figsize=(8, 8))

plt.scatter(data[:50, 2], data[:50, 3], c='r', label='Iris-setosa')

plt.scatter(data[50:100, 2], data[50:100, 3], c='g',label='Iris-versicolor')

plt.scatter(data[100:, 2], data[100:, 3], c='b',label='Iris-virginica')
plt.xlabel('Petal length',fontsize=15)
plt.ylabel('Petal width',fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.title('Petal length vs. Petal width',fontsize=15)
plt.legend(prop={'size': 20})
plt.show()
```



Even when it comes to petal-length and petal-width, the above graph indicates a strong correlation for setosa flowers which are densely clustered together.

Next, to further validate the claim of how petal-length and petal-width are correlated, let's plot a correlation matrix for all the three species.

dataset.iloc[:,2:].corr()

	petal-length	**petal-width**
petal-length	1.000000	0.962865
petal-width	0.962865	1.000000

The above table signifies a strong correlation of 0.96 for petal-length and petal-

width when all three species are combined.

Let's also analyze the correlation between all three species separately.

dataset.iloc[:50,:].corr() #setosa

	sepal-length	sepal-width	petal-length	petal-width
sepal-length	1.000000	0.742547	0.267176	0.278098
sepal-width	0.742547	1.000000	0.177700	0.232752
petal-length	0.267176	0.177700	1.000000	0.331630
petal-width	0.278098	0.232752	0.331630	1.000000

dataset.iloc[50:100,:].corr() #versicolor

	sepal-length	sepal-width	petal-length	petal-width
sepal-length	1.000000	0.525911	0.754049	0.546461
sepal-width	0.525911	1.000000	0.560522	0.663999
petal-length	0.754049	0.560522	1.000000	0.786668
petal-width	0.546461	0.663999	0.786668	1.000000

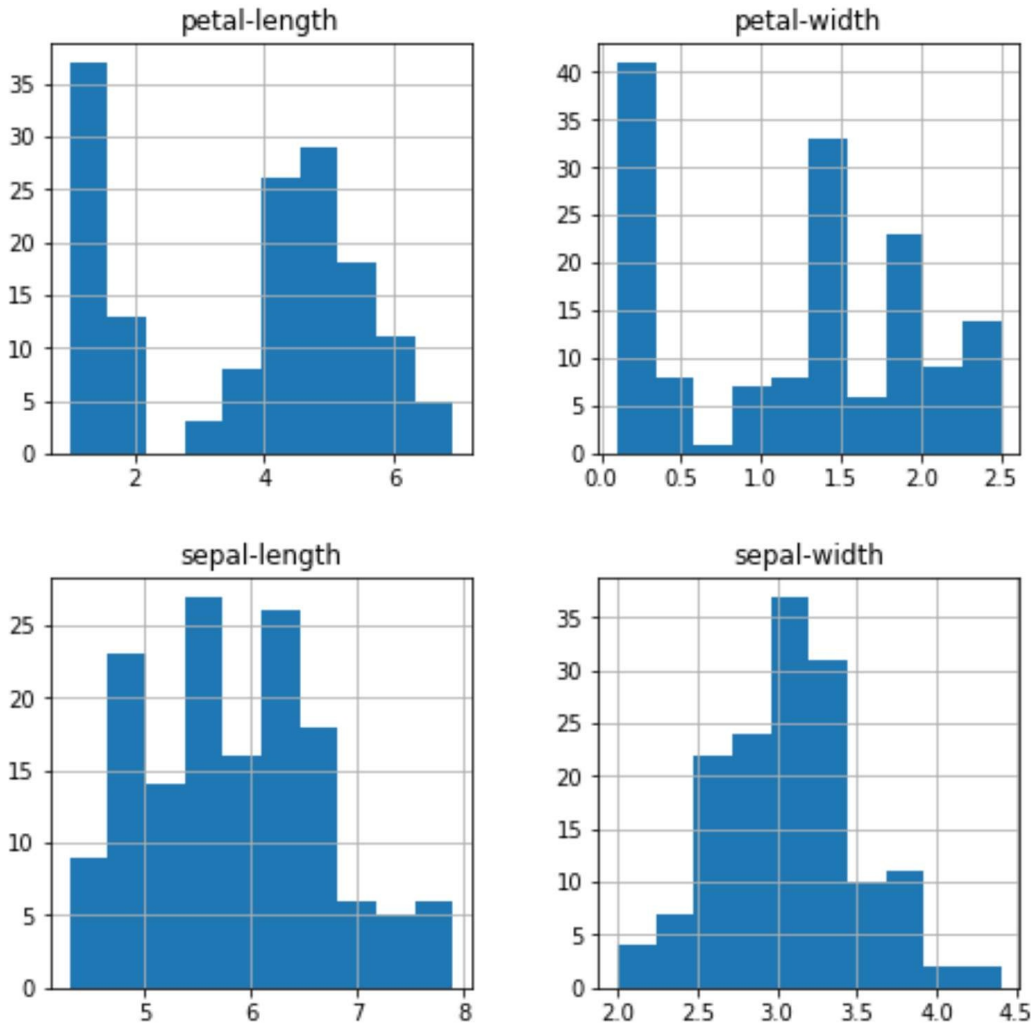
dataset.iloc[100:,:].corr() #virginica

	sepal-length	sepal-width	petal-length	petal-width
sepal-length	1.000000	0.457228	0.864225	0.281108
sepal-width	0.457228	1.000000	0.401045	0.537728
petal-length	0.864225	0.401045	1.000000	0.322108
petal-width	0.281108	0.537728	0.322108	1.000000

From the above three tables, it is pretty much clear that the correlation between petal-length and petal-width of setosa and virginica is 0.33 and 0.32 respectively. Whereas, for versicolor it is 0.78.

Next, let's visualize the feature distribution by plotting the histograms:

```
fig = plt.figure(figsize = (8,8))
ax = fig.gca()
dataset.hist(ax=ax)
plt.show()
```



The petal-width, petal-length, and sepal-length displays a unimodal distribution, whereas sepal-width shows a kind of Gaussian distribution. All these are useful analysis because then you can think of using an algorithm that works well with this kind of distribution.

Next, you will analyze whether all the four attributes are on the same scale or not; this is an essential aspect of ML. pandas data frame has an inbuilt function called describe that gives you the count, mean, max, min of the data in a tabular format.

dataset.describe()

	sepal-length	sepal-width	petal-length	petal-width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

You can see that all the four attributes have a similar scale between 0 & 8 and are in centimeters if you want you can further scale it down to between 0 and 1.

Even though you all know that there are 50 samples per class, i.e., ~33.3% of the total distribution, but still let's recheck it!

```
print(dataset.groupby('species').size())
species
Iris-setosa      50
Iris-versicolor  50
Iris-virginica   50
dtype: int64
```

Preprocessing your data

After having loaded the data and analyzed it extensively, it is time to prepare your data which you can then feed to your ML model. In this section, you will

preprocess your data in two ways: normalizing your data and splitting your data into training and testing sets.

Normalizing your data

There can be two ways by which you can normalize your data:

- Example normalization wherein you normalize each sample,
- Normalization of features where you normalize every feature in the same manner for all samples.

Now the question is why or when do you need to normalize your data? And do you need to standardize the Iris data?

Well, the answer is pretty much all the time. It is a good practice to normalize your data as it brings all the samples in the same scale and range. Normalizing the data is crucial when the data you have is not consistent. You can check for inconsistency by using the describe() function that you studied above which will give you max and min values. If the max and min values of one feature are significantly larger than the other feature then normalizing both the features to the same scale is very important.

Let's say X is one feature having a larger range and Y being the second feature with a smaller range. Then, the influence of feature Y can be overpowered by feature X's influence. In such a case, it becomes important to normalize both the features X and Y.

In Iris data, normalization is not required.

Let's print the describe() function again and see why you do not need any normalization.

dataset.describe()

	sepal-length	**sepal-width**	**petal-length**	**petal-width**
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333

std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

The values of all the characteristics are within the range of 0.1 and 7.9, which you can consider acceptable. Hence, you do not need to apply any normalization to the Iris dataset.

Splitting the data

This is another significant aspect of ML since your goal is to make a model capable enough to be able to take decisions or classify data in a test environment without any human intervention. Hence, before deploying your ML model in the industry, you need to make sure that the model can generalize well on the testing data.

For this purpose, you need a training and testing set. Coming back to the Iris data, you have 150 samples, you will be training your ML model on 80% of the data and the remaining 20% of the data will be used for testing.

In data-science, you will often come across a term called Overfitting which means that your model has learned the training data very well but fails to perform on the testing data. So, splitting the data into training and testing or validation set will often help you to know whether your model is overfitting or not.

For training and testing set split, you will use the sklearn library which has an in-built splitting function called train_test_split. So, let's split the data.

```
from sklearn.model_selection import train_test_split
train_data,test_data,train_label,test_label = train_test_split(dataset.iloc[:,:3],
dataset.iloc[:,4], test_size=0.2, random_state=42)
```

Note that the random_state is a seed that takes a random_state as input if you change the number the split of the data will also change. However, if you keep the random_state same and run the cell multiple times the data splitting will remain unchanged.

Let's quickly print the shape of training and testing data along with its labels.

```
train_data.shape,train_label.shape,test_data.shape,test_label.shape
((120, 3), (120,), (30, 3), (30,))
```

Finally, it's time to feed the data to the k-nearest neighbor algorithm!

The KNN Model

After all the loading, analyzing and preprocessing of the data, it is now time when you will feed the data into the KNN model. To do this, you will use sklearn's inbuilt function neighbors which has a class called KNeigborsClassifier in it.

Let's start by importing the classifier.

from sklearn.neighbors import KNeighborsClassifier

In order to decide the best value for hyperparameter k, you will do something called grid-search. You will train and test your model on 10 different k values and finally use the one that gives you the best results.

Let us initialize a variable neighbors(k) which will have values ranging from 1-9 and two numpy zero matrices namely train_accuracy and test_accuracy each for training and testing accuracy. You will need them later to plot a graph to choose the best neighbor value.

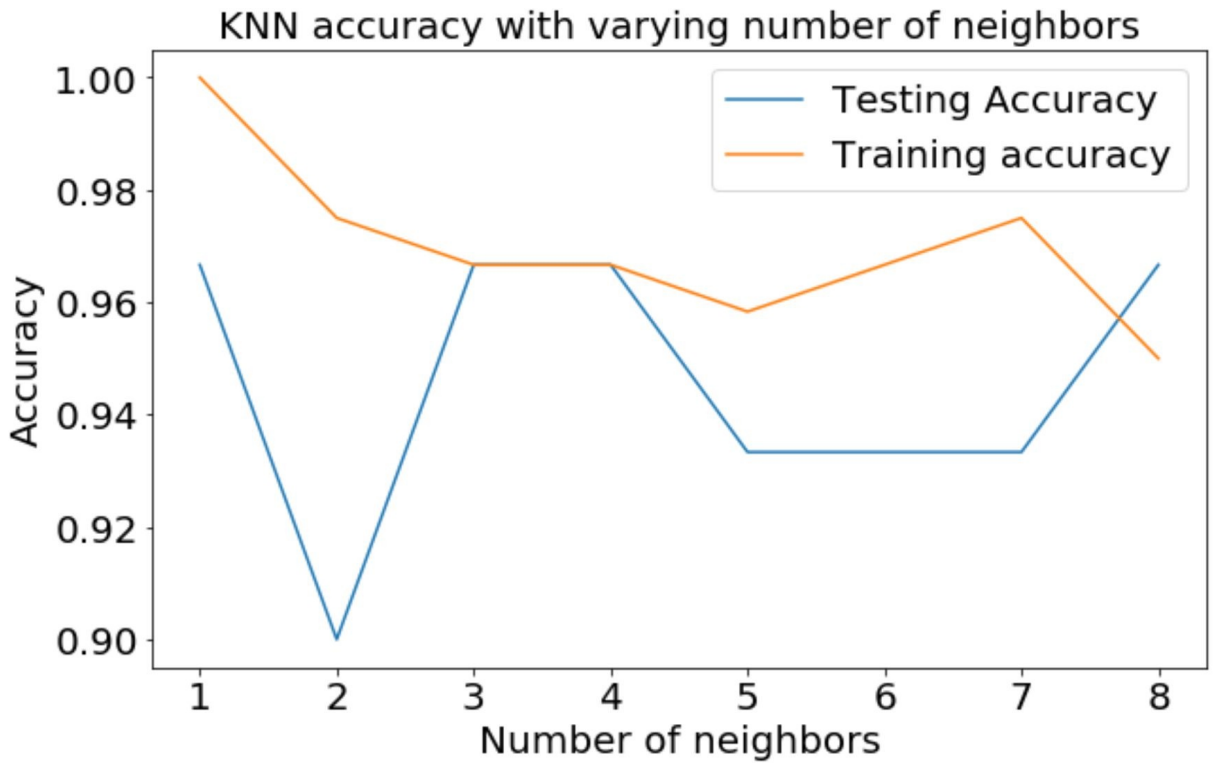
```
neighbors = np.arange(1,9)
train_accuracy =np.zeros(len(neighbors))
test_accuracy = np.zeros(len(neighbors))
```

Next piece of code is where all the magic will happen. You will enumerate over all the nine neighbor values and for each neighbor you will then predict both on training and testing data. Finally, store the accuracy in the train_accuracy and test_accuracy numpy arrays.

```
for i,k in enumerate(neighbors):
    knn = KNeighborsClassifier(n_neighbors=k)

    #Fit the model
    knn.fit(train_data, train_label)

    #Compute accuracy on the training set
    train_accuracy[i] = knn.score(train_data, train_label)

    #Compute accuracy on the test set
    test_accuracy[i] = knn.score(test_data, test_label)
```

Next, you will plot the training and testing accuracy using matplotlib, with accuracy vs. varying number of neighbors graph you will be able to choose the best k value at which your model performs the best.

```
plt.figure(figsize=(10,6))
plt.title('KNN accuracy with varying number of neighbors',fontsize=20)
plt.plot(neighbors, test_accuracy, label='Testing Accuracy')
plt.plot(neighbors, train_accuracy, label='Training accuracy')
plt.legend(prop={'size': 20})
plt.xlabel('Number of neighbors',fontsize=20)
plt.ylabel('Accuracy',fontsize=20)
plt.xticks(fontsize=20)
plt.yticks(fontsize=20)
plt.show()
```



Well, by looking at the above graph, it looks like when n_neighbors=3, both the model performs the best. So, let's stick with n_neighbors=3 and re-run the training once again.

```
knn = KNeighborsClassifier(n_neighbors=3)

#Fit the model
knn.fit(train_data, train_label)

#Compute accuracy on the training set
train_accuracy = knn.score(train_data, train_label)

#Compute accuracy on the test set
test_accuracy = knn.score(test_data, test_label)
```

Evaluating your Model

In the last segment of this tutorial, you will be evaluating your model on the testing data using a couple of techniques like confusion_matrix and

classification_report.

Let's first check the accuracy of the model on the testing data.

test_accuracy
0.9666666666666667

Viola! It looks like the model was able to classify 96.66% of the testing data correctly. Isn't that amazing? With few lines of Python program, you were able to train an ML model that is now able to tell you the flower name by using only four features with 96.66% accuracy. Who knows maybe it performed way better than a human can.

Confusion Matrix

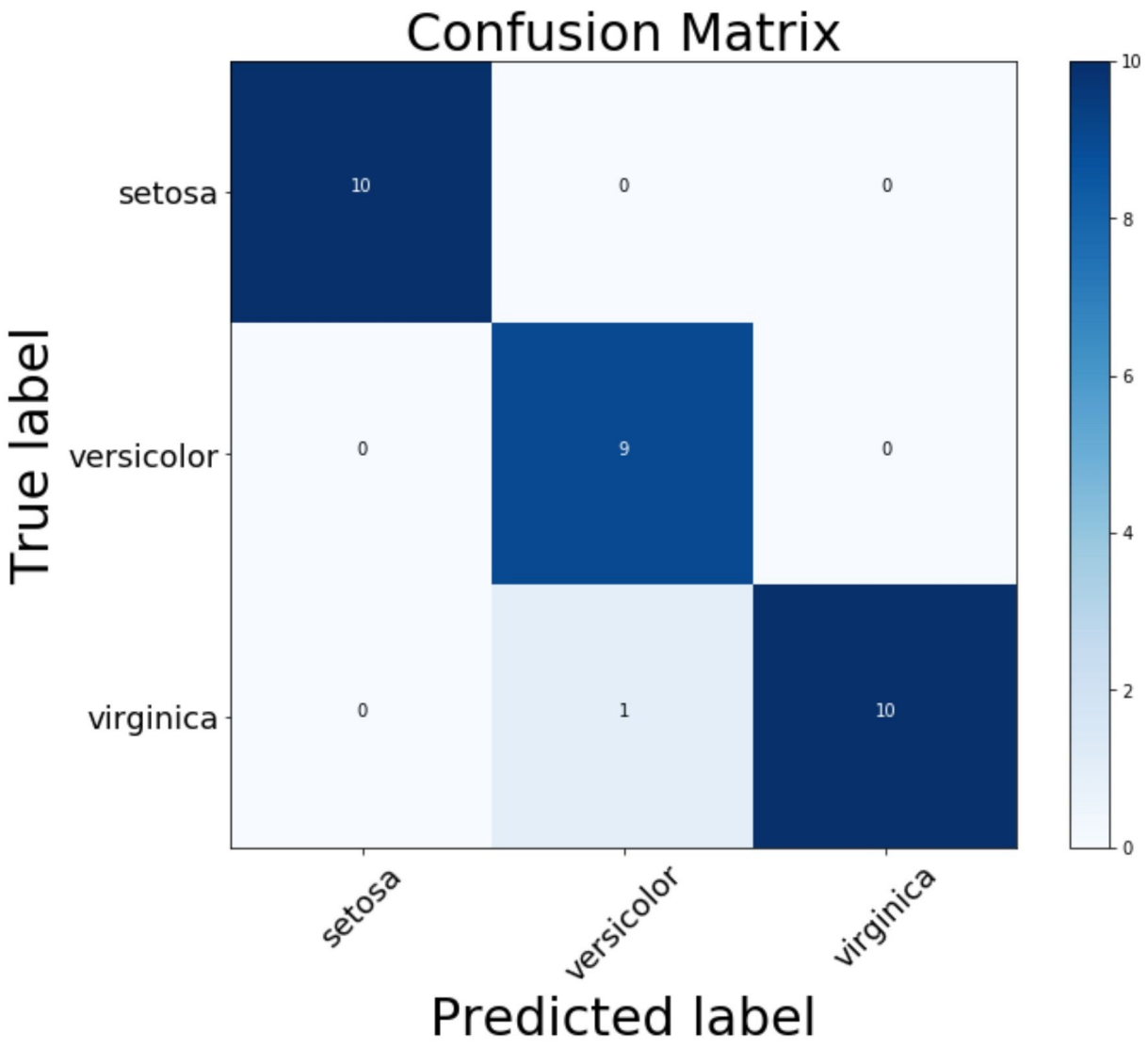
A confusion matrix is majorly used to show the functioning of your model on the data tested for whose true information or labels are already known.

Scikit-learn provides a function that calculates the confusion matrix for you.

prediction = knn.predict(test_data)

The following plot_confusion_matrix() function has been modified and acquired from this source:

```
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):


    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label',fontsize=30)
    plt.xlabel('Predicted label',fontsize=30)
    plt.tight_layout()
    plt.xticks(fontsize=18)
    plt.yticks(fontsize=18)

class_names = load_iris().target_names


# Compute confusion matrix
cnf_matrix = confusion_matrix(test_label, prediction)
np.set_printoptions(precision=2)

# Plot non-normalized confusion matrix
plt.figure(figsize=(10,8))
plot_confusion_matrix(cnf_matrix, classes=class_names)
plt.title('Confusion Matrix',fontsize=30)
plt.show()
```



From the above confusion_matrix plot, you can observe that the model classified all the flowers correctly except the one versicolor flower which is classified as a virginica flower.

Report for Classification

It assists you to identify the wrongly classified classes in a detailed way by giving accuracy, recall and F1 score for every class. Most of the time you will make use of the sklearn library to visualize the report.

```
from sklearn.metrics import classification_report
print(classification_report(test_label, prediction))
```

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	10
Iris-versicolor	0.90	1.00	0.95	9
Iris-virginica	1.00	0.91	0.95	11
micro avg	0.97	0.97	0.97	30
macro avg	0.97	0.97	0.97	30
weighted avg	0.97	0.97	0.97	30

Go Further!

First of all congratulations to all those who successfully made it till the end! But this was just the start. There is still a long way to go!

This tutorial majorly dealt with the basics of ML and the implementation of one kind of ML algorithm known as KNN with Python. The Iris data set that you used was pretty small and a little simple.

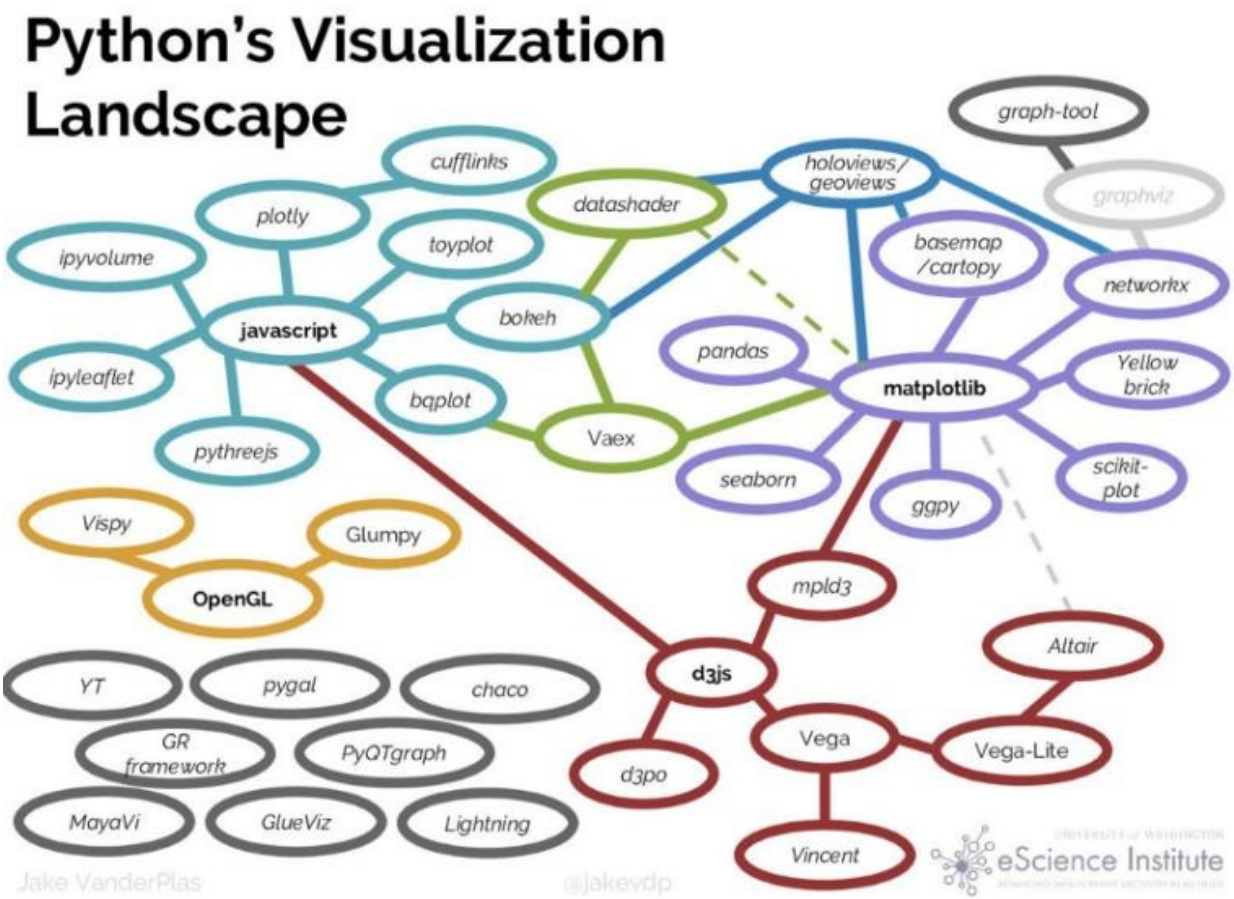
If this tutorial ignited an interest in you to learn more, you can try using some other datasets or try learning about some more ML algorithms and maybe apply on the Iris dataset to observe the effect on the accuracy. This way you will learn a lot more than just understanding the theory!

Chapter 5: Data Processing, Analysis, and Visualizations

If you work with data, then Data Visualization is a vital part of your daily routine. And if you use Python for your analysis, you ought to be overwhelmed by the sheer amount of choices present in the form of Data Visualization libraries. Some libraries like Matplotlib are used for initial basic exploration but are not so useful for showing complex relationships in data. There are some which work well with large datasets while there are still others which mainly focus on 3D renderings. There isn't a single visualization library which can be referred to as the best one. There are certain features in one that is better than the other and vice versa. In short, there are a lot of options, and it is impossible to learn and try them all or maybe, get them all to work together. So how do we get our job done? PyViz may have an answer

Python's Current Visualization Landscape

The existing Python Data Visualization system appears to be a confusing Mesh.

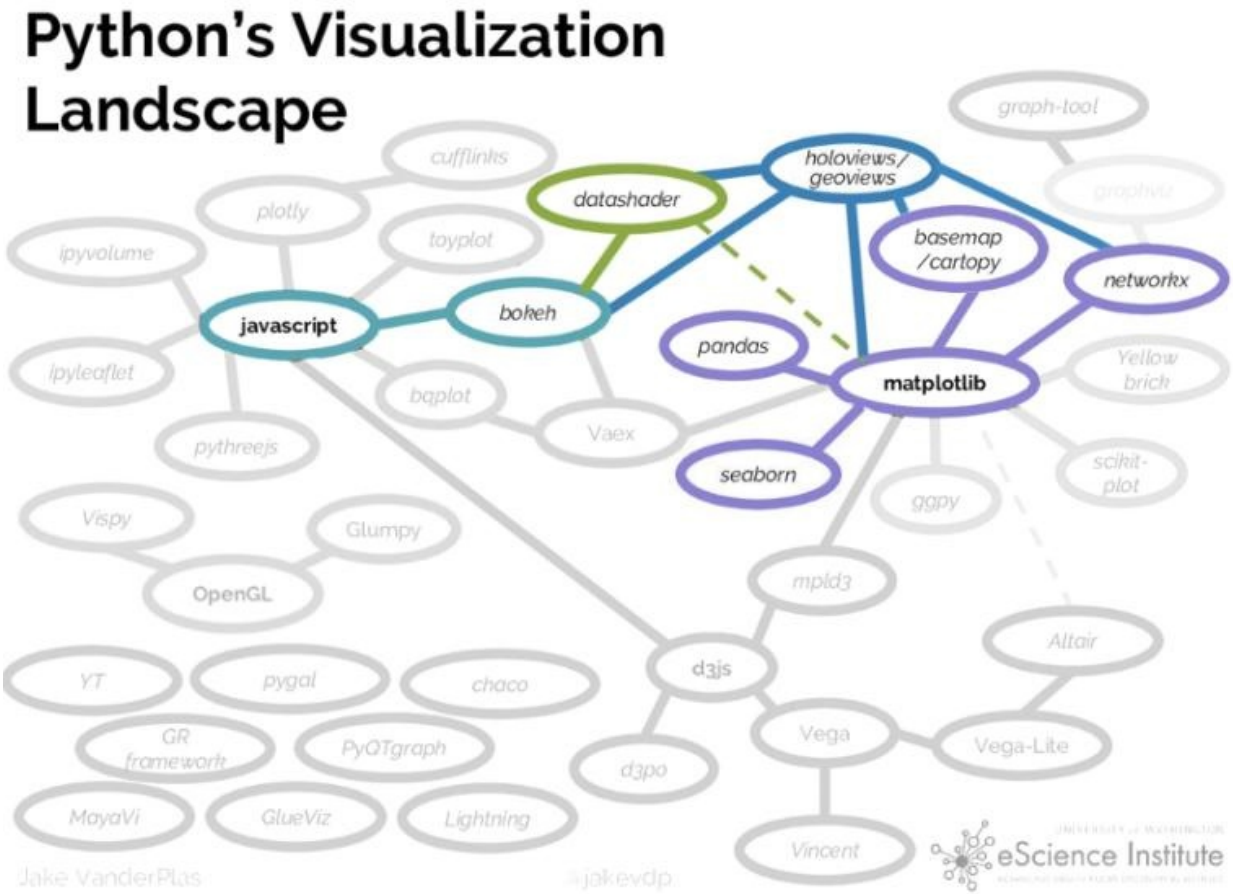


Now, to choose the best tool for our job from amongst all of these is a bit tricky and confusing. PyViz tries to plug this situation. It helps in streamlining the work of using large and small sets of data in an internet browser, whether building dashboards, creating simple widget-based tools, or accomplishing exploratory analysis.

PyViz Ecosystem

PyViz is defined as a coordinated and monitored effort to ensure visualization of data in Python is effective, efficient and more powerful. PyViz is made up of a set of open-source packages of Python to work suitably with both small and large datasets right in the web browsers. PyViz is just the choice for something as simple as mere EDA, or something as complex as creating a widget enabled dashboard.

Here is Python's visualization landscape with PyViz.



PyViz Goals

Some of the important goals of PyViz are:

- Emphasis should be on data of any size not coding
- Full functionality and interactivity should be available right in the browsers(not desktops)
- The focus should be more on people who are Python users and not web programmers.
- Again should be more on 2D viz more than 3D.
- The exploitation of general -purpose SciPy/PyData tools with which the Python users are already familiar.

Apart from this, PyViz core tools can work seamlessly with libraries such as HoloViews and Plotly.

Resources

PyViz provides examples, demos and training materials documenting how to solve visualization problems. This tutorial provides starting points for solving your visualization problems. The entire tutorial material is also hosted at their Github Repository.

PyViz Tutorial — PyViz 0.9.13a3 documentation

How to solve visualization problems with Python tools.pyviz.org

Once everything is installed, the following cell should print '1.11.0a4' or later:

```
import holoviews as hv
hv.__version__
'1.11.0a11'
hv.extension('bokeh', 'matplotlib')
#should see the HoloViews, Bokeh, and Matplotlib logos
#Import necessary libraries
import pandas
import datashader
import dask
import geoviews
import bokeh
```

If it completes without errors, your environment should be ready to go.

Exploring Data with PyViz

In this section, we will see how different libraries are effective in bringing out different insights from data, and their conjunction can help to analyze data in a better way.

Dataset

The dataset being used pertains to the number of cases of measles and pertussis recorded per, 100,000 people over time in each state of the US. The dataset comes pre-installed with the PyViz tutorial.

Data Exploration with Pandas

In any Data Science project, it is but natural to begin the exploration with pandas. Let us import and display the first few rows of our dataset.

import pandas as pd

diseases_data = PD.read_csv('../data/diseases.csv.gz')

diseases_data.head()

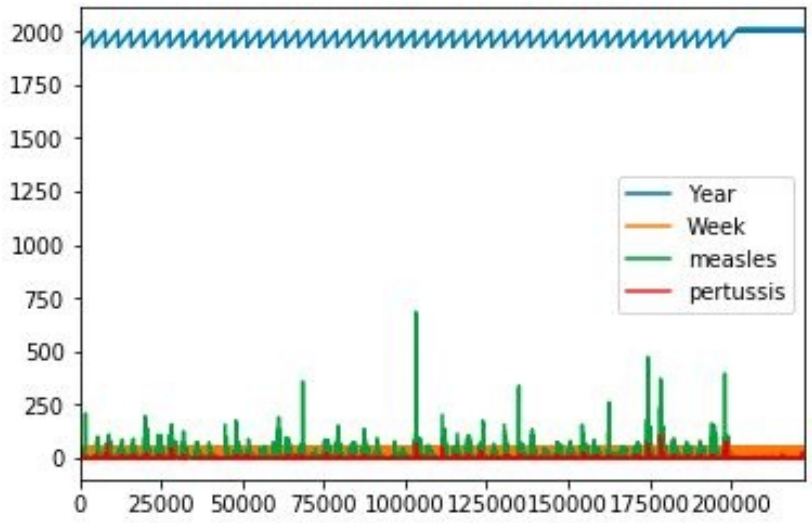
	Year	Week	State	measles	pertussis
0	1928	1	Alabama	3.67	NaN
1	1928	2	Alabama	6.25	NaN
2	1928	3	Alabama	7.95	NaN
3	1928	4	Alabama	12.58	NaN
4	1928	5	Alabama	8.03	NaN

Numbers are good, but a plot would give us a better idea about the patterns in the data.

Data Exploration with Matplotlib

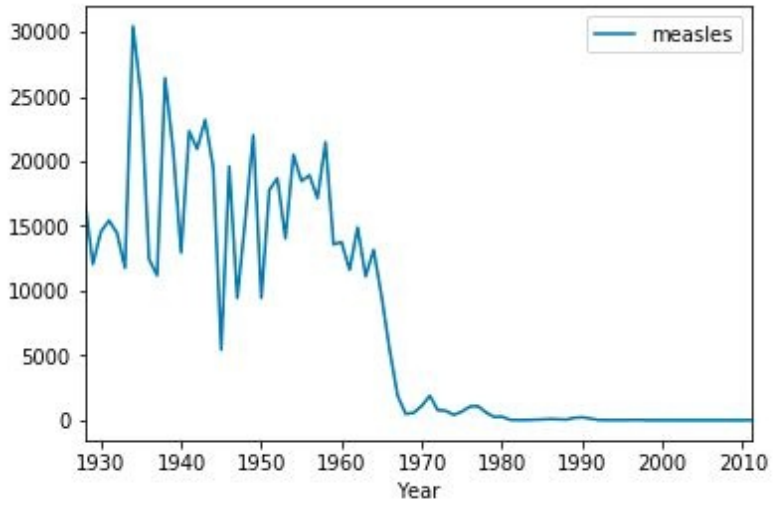
%matplotlib inline

diseases_data.plot();



This doesn’t convey much. Let’s do some manipulations with pandas to get meaningful results.

```
import numpy as np
diseases_by_year =
diseases_data[["Year","measles"]].groupby("Year").aggregate(np.sum)
diseases_by_year.plot();
```



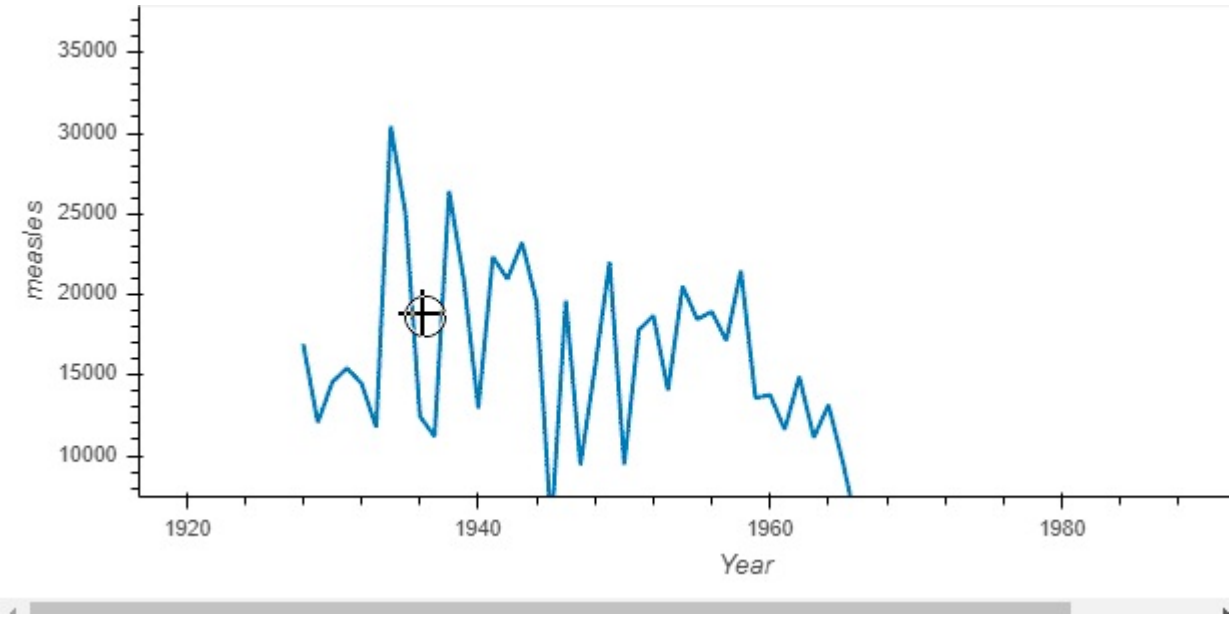
This makes much more sense. Here we can infer that around 1970, something happened which brought down the rate of measles to almost nil. This is true since measles vaccines were introduced in the US around the year 1963.

Data Exploration with HVPlot and Bokeh

The plots above convey the right information but provide no interactivity. This is because they are static plots without the functionalities of the pan, hover or zoom in a web browser. However, we can achieve this interactive functionality by a mere import of the **hvplot** package.

Import hvplot.pandas

diseases_by_year.hvplot()



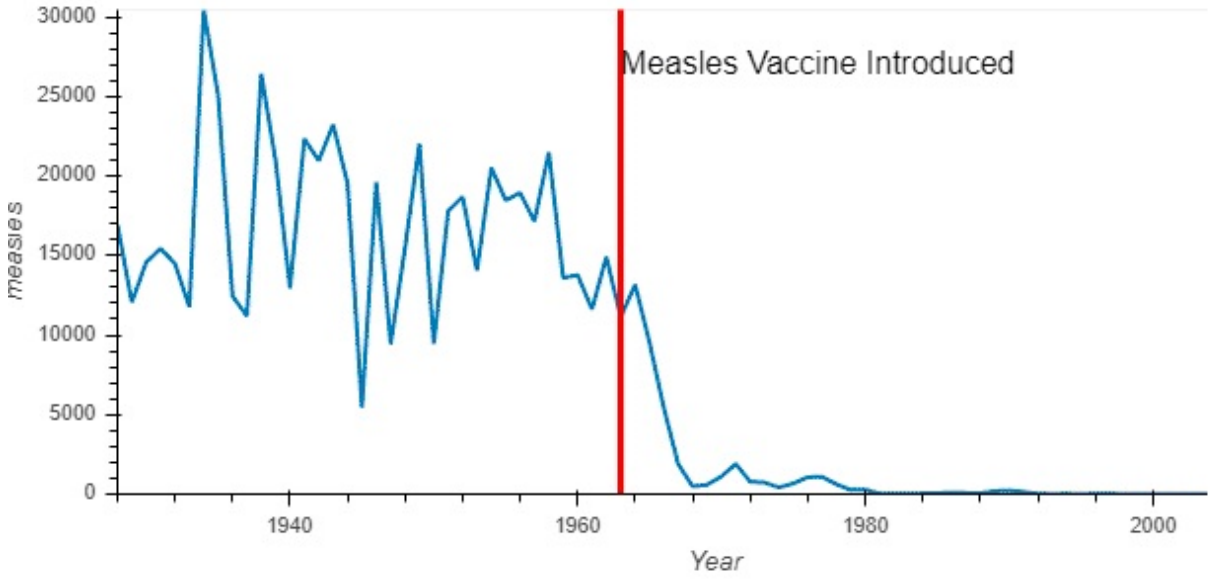
The outcome that is displayed by the call is known as a **HoloViews** object(HoloViews curve)which is displayed as a Bokeh curve or plot. HoloViews curves or plots are richer and easy to capture your comprehension while exploring the information.

Let us view what more can be done using HoloViews:

Capturing important points on the Plot itself

1963 was important concerning measles and how about we record this point on the graph itself. This will also help us to compare the number of measles cases before and after the vaccine introduction.

```
import holoviews as hv
vline = hv.VLine(1963).options(color='red')
vaccination_introduced = diseases_by_year.hvplot() * vline * \
   Hv.Text(1963, 27000, "Measles Vaccine Introduced", halign='left')
vaccination_introduced
```



Holoviews objects preserve the original data as opposed to other plotting libraries. For instance, it is possible to access the original data in a tabular format.

print(vaccination_introduced)

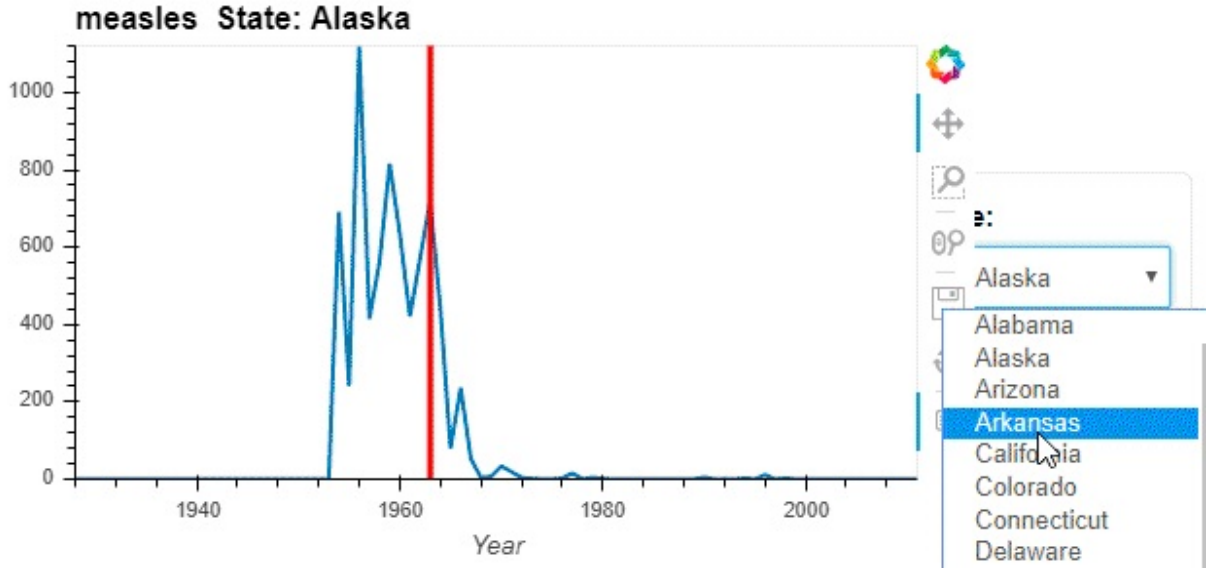
vaccination_introduced.Curve.I.data.head()

```
:Overlay
   .Curve.I :Curve   [Year]   (measles)
   .VLine.I :VLine   [x,y]
   .Text.I  :Text   [x,y]
```

	Year	measles
0	1928	16924.34
1	1929	12060.96
2	1930	14575.11
3	1931	15427.67
4	1932	14481.11

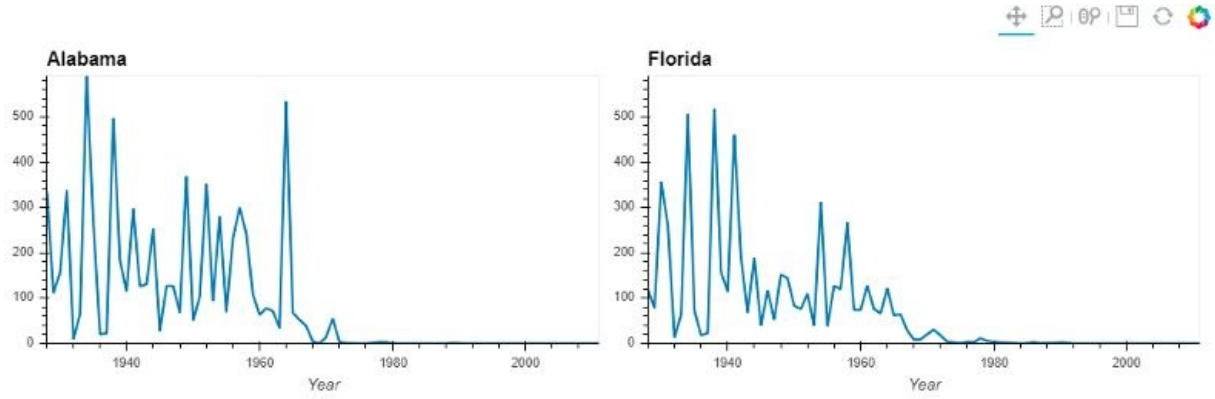
Here we were able to use the data that was used for making the plot. Also, it is now very easy to break data in many different ways as shown in the code snippet below.

```
measles_agg = diseases_data.groupby(['Year', 'State'])['measles'].sum()
by_state = measles_agg.hvplot('Year', groupby='State', width=500,
dynamic=False)
by_state * vline
```



Instead of a dropdown, we can place charts side by side for better comparison.

```
by_state["Alabama"].relabel('Alabama') + by_state["Florida"].relabel('Florida')
```



We can also change the type of plots, say to a bar chart. Let us compare the measles pattern from 1980 to 1985 across four states.

```
states = ['New York', 'Alabama', 'California', 'Florida']
measles_agg.loc[1980:1990, states].hvplot.bar('Year', by='State', rot=90)
```

It is quite evident from the examples above that by choosing **HoloViews+Bokeh**

plots; we get the ability to explore data in our browser itself, with full interactivity and minimal code.

Visualizing large datasets with PyViz

PyViz also enables working on very large datasets with ease. For such datasets, other members of PyViz suite come into the picture.

- GeoViews
- Datashader
- Panel
- Param
- Colorcet for perceptually uniform colormaps for big data

To show you the capabilities of these libraries when handling the voluminous amount of data, let's work with the NYC taxi dataset which consists of data about a whopping 10 million taxi trips. Again this data is already provided in this book.

We can also add widgets to control the selections. This can be either done in the notebook or a standalone server by marking the servable objects with .servable() then running the .ipynb file through Bokeh Server or extracting the code to a separate .py file and typing the following code snippet:

```
import param, the panel as pn
from colorcet import palette
class NYCTaxi(param.Parameterized):
   alpha = param.Magnitude(default=0.75, doc="Map tile opacity")
   cmap = param.ObjectSelector('fire',
objects=['fire','bgy','bgyw','bmy','gray','kbc'])
   location = param.ObjectSelector(default='dropoff', objects=['dropoff',
'pickup'])
   def make_view(self, **kwargs):
      pts   = hv.Points(taxi, [self.location+'_x', self.location+'_y'])
      trips = datashade(pts, cmap=palette[self.cmap], **dopts)
      return tiles.options(alpha=self.alpha) * trips
explorer = NYCTaxi(name="Taxi explorer")
pn.Row(explorer.param, explorer.make_view).servable()
```

Taxi explorer

The interactiveness does not render on GitHub, but you can still access the notebook and run locally.

Conclusion

The PyViz tools help us to create beautiful visualizations even with a small amount of code. The PyViz tools bundle up together to create deployable, flexible, scalable, and high-performance dashboards, applications, and visualizations, without using web technologies such as JavaScript. This book is just a mere introduction to the multi useful PyViz ecosystem. Go through the entire book to understand the intricacies and its usage for different types of data.

Chapter 6: Case Studies

The major shift that has transformed application development has begun remolding Information Technology operations. To keep pace with digital transformation, Information Technology Operations is changing how it manages its ecosystem, turning to artificial intelligence, analytics, and Machine Learning.

Cloud, agile, and DevOps are making new demands that Information Technology can meet only through the use of Artificial Intelligence and its ilk. Professionals who wish to advance their Information Technology Operations career in the future will need to get a handle on Artificial Intelligence Operations, the term used to describe what it takes for Information Technology Operations to handle the digital transformation.

One way to get the knowledge needed to advance in the coming brave new world of Information Technology Operations is to attend conferences where experts are discussing Artificial Intelligence, analytics, and machine-learning developments and trends, and where you can consult with your peers about their experience with Artificial Intelligence Operations.

This year people will not only witness big advancements in ML but also unimaginable steps and new applications that will force people to go back to square one and re-consider whether ML is a bane or a boon. I like to imagine that it may be just like any other advancement in technology – it is about how you use it. That being said let us dive into the upcoming applications of Artificial Intelligence and ML this year. These applications are mostly in R&D stages and about to be out to the public, or are out to the public but in beta stages, and again to get an official flag off.

- **AI chips**

Remember when you used to play those arcade PC games before you moved on

to consoles like PS4 and Xbox? When playing low-end games, the CPU (usually Intel or AMD) was enough, but when playing the latest high-end games like Tomb Raider and Assasin's Creed Odyssey, you would need a dedicated GPU (or graphics card in layman's term). These are dedicated chips that boost only your graphics experience. Why am I talking about this? Well, a new dedicated chip is about to hit the market.

AI enabled chips have already debuted in a few cell-phones like iPhone XR, Huawei Honor Play, and the Galaxy Note 8. In these cell-phones, a dedicated AI chip does the heavy lifting during functionalities like ML-based voice assistant, AI-powered smart camera and more. This way, the AI chip decreases the workload of the primary processor and keeps the phone from lagging. But AI enabled chips in cell-phones are just the tip of the iceberg.

Google, Microsoft, Amazon, Intel, and Facebook are all investing heavily in dedicated infrastructure that will boost AI-based functionalities, be it a chip that you can buy and plugin or a cloud-based CPU that can be used for ML and neural networks. The latest in the news among these are the Intel Nervana (developed by Intel, where Facebook is a significant development partner), Google Cloud TPU, Microsoft's project Brainwave, and Amazon's AWS Inferentia.

Most of the mentioned projects are not yet publicly available or are still in a Beta stage. Hence, 2019 could be the year when these projects will see the light of the day, and more powerful AI chips will make real-time AI-based applications a reality.

- **The marriage of IOT and AI**

A lot has happened in both the domains, but the proper implementation of an IoT along with AI enabled solution is yet to be seen. The most popular use case that is stated on this topic is capturing data from cars using sensors and using the collected data to decide on an insurance amount. But honestly speaking how

many insurance companies have used that method to calculate your premium amount?

Things are about to change in 2019, with the emergence of AI chips, cheaper and newer cloud-based AI services, and more prominent companies investing in IoT and AI. Many companies are going to bring out their self-driving cars, and CES. 2019 gave a glimpse into how well IOT and AI together have helped companies dream bigger.

- **More applications of Automated ML**

While ML is a term most are familiar with; Automated ML is something that one might say is still in Beta stage. Automated ML refers to solving repetitive tasks through automation by applying ML to ML itself. For example, say one has to clean the data before he can apply different algorithms and see which algorithm fits the data best. The data cleaning part takes up a lot of time, and studies have shown, that it is a manual process that scientists loathe to repeat with new data sets again and again. Processes like these could be solved with the help of AutoML and 2019 is going to see a lot more applications as predicted by many.

- **AI will automate DevOps**

IT services and infrastructures are a headache that you have to take on to keep your business running smoothly. However, various acts that are performed by DevOps personnel can be automated, such as monitoring, debugging, scaling up, and more. DevOps will slowly give way to AIOps, and this will make it easier for everyday coders to take care of procedures and pipelines without the need to take the headache of setups and maintenance. Various papers have come up that predict the future of DevOps through AIOps.

- **Personalized medicines**

Do you take the same pills for sleeping that your neighbor uses? Certainly, the dosage might not be enough for you. Or one of the ingredients might not be good for you. Or the pill might not have any effect on you due to certain reasons. Why

not have medicines tailored specifically for you? Personalized Medicines follow a procedure – a patient is analyzed, along with his or her lifestyles and habits and only then a pill of specific composition is recommended. With the help of the 100,000 Genomes Project, Personalized Medicines are expected to see huge growth in 2019, even though they might not be available publicly yet. Several big conferences like the 11th International Conference on Personalized, Preventive and Predictive, Preventive Molecular and Medicine Diagnostics, are also scheduled this year.

ML will help analyze patient data to decide which medicine would suit one best, leaving behind the age-old "one size fits all" theory.

- **Further evolution of ML-based assistants**

In 2018, we were all blown away by the new and advanced Google Assistant. What did not happen is its commercial application. But that could be seen in 2019. Many voice assistants like Alexa have been gathering a lot of data from users, and 2019 could see these assistants growing much more intelligent and human-like. Voice assistants are supposed to move on from your living room to public places, your cars, announcement systems, ATMs, and more. Many are betting that 2019 will be the ***Year of Voice Assistants***.

- **Better metrics from machines using AI**

How good would it be if you could tell which machine in your factory is almost worn out and needs repair before it breaks down and causes loss of work for a day?

Technical and industrial equipment and machines are usually serviced using a fixed schedule. This results in wasted labor and the risk of sudden and unexpected equipment failures from time to time. Once sensors are used with these machines and the data collected are fed to ML models, we can get better performance and more efficient servicing schedules.

An implementation of the following is GE's creation of ***Digital Twins*** where a

virtual model is built for big machines. Hundreds of sensors are used to make sure that the twin is updated with the condition of the real machine and this way the currently deployed 650,000 twins are managed and serviced. Further implementations of digital twins are expected to be implemented across intelligent factories across the globe this year.

8. Computer vision will shape the future of surveillance

Computer vision and surveillance have become a necessity in places like malls, airports or even in city crossroads. Many technology companies have been quietly implementing their software in multiple places around the world to spot, track and report situations as and when necessary. But things need not go that far. Wouldn't it be amazing if our home camera spotted an unknown man trying to break in and Google Assistant alerted us about it?

Companies like Google are heavily investing in computer vision (remember how it asks you to mark images with cars or signboards sometimes when you are trying to open a website?), and 2019 would see more players in the market. Surveillance systems are about to get a lot faster, and surveillance is soon going to become not just a video stream but rather work as a stimulus to automated responses as well, based on the ground situation.ß

9. Healthcare will see further AI implementations

Healthcare has evolved over the years, and ML has brought in more changes than just the rise of personalized medicines. Recently China claimed to have created the first Gene-edited babies. The tool used called CRISPR-CAS9, is supposed to operate on DNA to supply a required gene or disable the one that is causing disease or illness. Although the act has not been published in peer-reviewed journals and we are yet to verify it, artificial intelligence is already being used to predict genomic changes resulting in cancer.

This year will see newer applications of gene modifications using AI to combat newer diseases and optimize humans. However, the ethical and legal dilemmas

and arguments behind such medical miracles remain a point of debate in the scientific world.

10. Social Credit Systems

A relatively new application of ML that has been implemented in China- it is a vast ranking system which will be monitoring the behavior of all the citizens using surveillance cameras and online activity monitors, to rank them based on their “social credit.“

Although it is supposed to be fully up and running by 2020, its implementation has already taken the country by storm, and people are being marked up or down depending on acts like smoking in public, boarding a train without a ticket or donating to charity. The exact methodology is unknown, but hopefully, a robust ML model uses data from various sources to decide whether to decrease a person’s score, increase it, or let it remain stagnant.

What sounds like a dystopian thought and seems like an episode from Black Mirror, is the reality of millions and we might see the implementation of the system covering all of China by the end of this year. And again, although the application is very new, the philosophy beyond it is argued by many.

Conclusion

Thank you for making it to the end of the book *Python Machine Learning for Beginners* we hope that you found it helpful and informative. Every effort was put to ensure that all the chapters give you valuable information. We intentionally used simple language to make sure that every person reading it gets empowered. The book has deliberately avoided complicated theories and stuck to simple practices that one can use at their conveniences when studying machine learning.

The moment you understand programming basics using Python language, it becomes easier to learn machine learning concepts. Machine Learning is essential in everyday life. This book has taken you through many concepts of machine learning. There is no one specific thing that a person can do to learn machine learning overnight. However, if you follow the right steps with dedication and commitment, you will get the results you seek. Combine a number of practical sessions to improve your python programming skills. If you are working with an experienced Python programmer, follow all the instructions he/she gives you and keep an open communication channel. Get dirty with the IDE and the keyboard.

The next step is to stop reading and start applying the lessons in real life. Do whatever you have identified as necessary to improve machine learning in real life. You will find that many people are still ignorant about the proper ways of enhancing artificial intelligence and other concepts of machine learning. You will realize that the majority of those who seem to have it all together lack the basic Python programming skills. To that end, try to engage them and teach them a thing or two you have learned herein. You may even recommend or gift this book to them.

You might also need to refer to this book at a later date. Keep it and review it as

often as you want. Just because you have reached the end of the book does not mean that there is nothing else to learn about Python and Machine Learning. Read more and expand your horizons. It is the only way you will achieve the mastery you seek. Pay attention to Artificial Intelligence particularly because without it Machine Learning is impossible. Use some of the tips herein to make the world a better place by coming up with your own inventions especially in Artificial Intelligence.

Finally, if you found this book useful in any way, a review on Amazon will be appreciated!