**The Python Machine Learning Eco-System**

1. We have a very diverse and varied support for Machine Learning in terms of programming languages and frameworks. There are Machine Learning libraries for almost all popular languages including C++, R, Julia, Scala, Python, etc.

🡪In this chapter we try to justify why Python is an apt language for Machine Learning. Once we have argued our selection logically, we give you a brief introduction to the Python Machine Learning (ML) eco-system.

🡪This Python ML ecosystem is a collection of libraries that enable the developers to extract and transform data, perform data wrangling operations, apply existing robust Machine Learning algorithms and also develop custom algorithms easily.

🡪These libraries include numpy, scipy, pandas, scikit-learn, statsmodels, tensorflow, keras, and so on. We cover several of these libraries in a nutshell so that the user will have some familiarity with the basics of each of these libraries. These will be used extensively in the later chapters of the book. An important thing to keep in mind here is that the purpose of this chapter is to acquaint you with the diverse set of frameworks and libraries in the Python ML ecosystem to get an idea of what can be leveraged to solve Machine Learning problems.

1. Python : An Introduction

🡪Python was created by Guido van Rossum at Stichting Mathematisch

Centrum in the Netherlands.

🡪The first version of Python was released in 1991.

🡪Guido wrote Python as a successor of the language called ABC.

🡪In the following years Python has developed into an extensively used high level language and a general programming language.

🡪Python is an interpreted language, which means that the source code of a Python program is converted into bytecode, which is then executed by the Python virtual machine.

🡪Python is different from major compiled languages like C and C++ as Python code is not required to be built and linked like code for these languages. This distinction makes for two important points:

* **Python code is fast to develop** : As the code is not required to be compiled and built, Python code can be much readily changed and executed. This makes for a fast development cycle.
* **Python code is not as fast in execution** : Since the code is not directly compiled and executed and an additional layer of the Python virtual machine is responsible for execution, Python code runs a little slow as compared to conventional languages like C, C++, etc.

🡺**Strengths Of Python** : Recently several surveys depicted Python to be the most popular language for Machine Learning and Data Science! We will compile a brief list of advantages that Python offers that probably explains its popularity.

1. **Easy to learn** : Python is a relatively easy-to-learn language. Its syntax is simple for a beginner to learn and understand. When compared with languages likes C or Java, there is minimal boilerplate code required in executing a Python program.
2. **Supports multiple programming paradigms** : Python is a multiparadigm, multi-purpose programming language. It supports object oriented programming, structured programming, functional programming, and even aspect oriented programming. This versatility allows it to be used by a multitude of programmers.
3. **Extensible** : Extensibility of Python is one of its most important characteristics. Python has a huge number of modules easily available which can be readily installed and used. These modules cover every aspect of programming from data access to implementation of popular algorithms. This easy-to-extend feature ensures that a Python developer is more productive as a large array of problems can be solved by available libraries.
4. **Active open source community** : Python is open source and supported by a large developer community. This makes it robust and adaptive. The bugs encountered are easily fixed by the Python community. Being open source, developers can tinker with the Python source code if their requirements call for it.

🡺Pitfalls Of Python :

🡪Although Python is a very popular programming language, it comes with its own share of pitfalls. One of the most important limitations it suffers is in terms of execution speed. Being an interpreted language, it is slow when compared to compiled languages. This limitation can be a bit restrictive in scenarios where extremely high performance code is required.

🡪Although we have to admit it can never be as fast as a compiled language, we are convinced that it makes up for this deficiency by being super-efficient and effective in other departments.

1. Setting Up Python Environment

🡪The starting step for our journey into the world of Data Science is the setup of our Python environment. We usually have two options for setting up our environment :

* Install Python and the necessary libraries individually
* Use a pre-packaged Python distribution that comes with necessary libraries, i.e. Anaconda

🡪Anaconda is a packaged compilation of Python along with a whole suite of a variety of libraries, including core libraries which are widely used in Data Science.

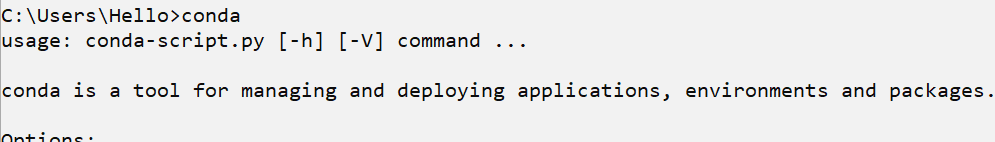
🡪Developed by Anaconda, formerly known as Continuum Analytics, it is often the go-to setup for data scientists. Travis Oliphant, primary contributor to both the numpy and scipy libraries, is Anaconda’s president and one of the co-founders.

🡪The Anaconda distribution is BSD licensed and hence it allows us to use it for commercial and redistribution purposes. Thus, we can get started with our Data Science journey with just one download and install. The Anaconda distribution is widely used across industry Data Science environments and it also comes with a wonderful IDE, Spyder (Scientific Python Development Environment), besides other useful utilities like jupyter notebooks, the IPython console, and the excellent package management tool, conda. Recently they have also talked extensively about Jupyterlab, the next generation UI for Project Jupyter.

🡺Set Up Anaconda Python Environment

🡪The first step in setting up your environment with the required Anaconda distribution is downloading the required installation package from https://www.anaconda.com/download/ , which is the provider of the Anaconda distribution.

🡪Installing the downloaded file is as simple as double-clicking the file and letting the installer take care of the entire process. To check if the installation was successful, just open a command prompt or terminal and write conda, And if there are many statements coming than it is installed successfully otherwise not, Example:



1. Why Python For Data science?

🡪Python has a lot of advantages that makes it a language of choice when it comes to the practices of Data Science, Some of them are as shown below:

* + **Powerful Set Of Packages** : Python is known for its extensive and powerful set of packages. In fact one of the philosophies shared by Python is batteries included, which means that Python has a rich and powerful set of packages ready to be used in a wide variety of domains and use cases.

🡪Packages like numpy, scipy, pandas, scikit-learn, etc., which are tailor-made for solving a variety of real-world Data Science problems, and are immensely powerful.

* + **Easy and Rapid Prototyping** : Using the REPL shell, IDEs, and notebooks, you can rapidly build and iterate over multiple research and development cycles and all the changes can be readily made and tested.
  + **Easy to collaborate** : Data science solutions are rarely a one man job. Often a lot of collaboration is required in a Data Science team to develop a great analytical solution. Luckily Python provides tools that make it extremely easy to collaborate for a diverse team.

🡪One of the most liked features, which empowers this collaboration, are jupyter notebooks. Notebooks are a novel concept that allow data scientists to share the code, data, and insightful results in a single place. This makes for an easily reproducible research tool.

* + **One Stop Solution** : Data Science as a field is interconnected to various domains. A typical project will have an iterative lifecycle that will involve data extraction, data manipulation, data analysis, feature engineering, modeling, evaluation, solution development, deployment, and continued updating of the solution.

🡪Python as a multipurpose programming language is extremely diverse and it allows developers to address all these assorted operations from a common platform. Using Python libraries you can consume data from a multitude of sources, apply different data wrangling operations to that data, apply Machine Learning algorithms on the processed data, and deploy the developed solution.

🡪This makes Python extremely useful as no interface is required, i.e. you don’t need to port any part of the whole pipeline to some different programming language.

🡪Also enterprise level Data Science projects often require interfacing with different programming languages, which is also achievable by using Python**. For example, suppose some enterprise uses a custom made Java library for some esoteric data manipulation, then you can use Jython implementation of Python to use that Java library without writing custom code for the interfacing layer.**

1. Introducing the python machine learning Ecosystem

🡪This section is structured to give you a gentle introduction and acquaint you with these core Data Science libraries.

🡪The list of components that we cover is by no means exhaustive but we have shortlisted them on the basis of their importance in the whole ecosystem.

**🡺Jupyter Notrebooks**

🡪Jupyter notebooks , formerly known as ipython notebooks, are an interactive computational environment that can be used to develop Python based Data Science analyses, which emphasize on reproducible research.

🡪The interactive environment is great for development and enables us to easily share the notebook and hence the code among peers who can replicate our research and analyses by themselves. These jupyter notebooks can contain code, text, images, output, etc., and can be arranged in a step by step manner to give a complete step by step illustration of the whole analysis process.

🡪This capability makes notebooks a valuable tool for reproducible analyses and research, especially when you want to share your work with a peer. While developing your analyses, you can document your thought process and capture the results as part of the notebook.

🡪This seamless intertwining of documentation, code, and results make jupyter notebooks a valuable tool for every data scientist.

🡪We will be using jupyter notebooks, which are installed by default with our Anaconda distribution. This is similar to the ipython shell with the difference that it can be used for different programming backends, i.e. not just Python. But the functionality is similar for both of these with the added advantage of displaying interactive visualizations and much more on jupyter notebooks.

🡺Installation and Execution

🡪We don’t require any additional installation for Jupyter notebooks, as it is already installed by the Anaconda distribution. We can invoke the jupyter notebook by executing the following command at the command prompt or terminal.



🡪This will start a notebook server at the address localhost:8888 of your machine. An important point to note here is that you access the notebook using a browser so you can even initiate it on a remote server and use it locally using techniques like ssh tunneling. This feature is extremely useful in case you have a powerful computing resource that you can only access remotely but lack a GUI for it. Jupyter notebook allows you to access those resources in a visually interactive shell. Once you invoke this command, you can navigate to the address localhost:8888 in your browser, to find the landing page depicted in Figure below, which can be used to access existing notebooks or create new ones.



🡪On the landing page we can initiate a new notebook by clicking the New button on top right. A notebook is just a collection of cells. There are three major types of cells in a notebook:

* 1. Code cells : Just like the name suggests, these are the cells that you can use to write your code and associated comments. The contents of these cells are sent to the kernel associated with the notebook and the computed outputs are displayed as the cells’ outputs.
  2. Markdown cells : Markdown can be used to intelligently notate the computation process. These can contain simple text comments, HTML tags, images, and even Latex equations. These will come in very handy when we are dealing with a new and non-standard algorithm and we also want to capture the stepwise math and logic related to the algorithm.
  3. Raw cells : These are the simplest of the cells and they display the text written in them as is. These can be used to add text that you don’t want to be converted by the conversion mechanism of the notebooks.

🡪Example,



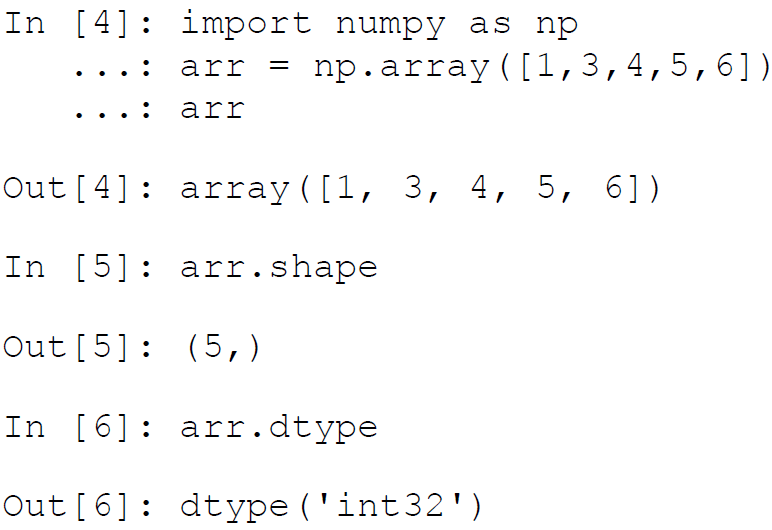
🡺Numpy

🡪Numpy is the backbone of Machine Learning in Python. It is one of the most important libraries in Python for numerical computations. It adds support to core Python for multi-dimensional arrays (and matrices) and fast vectorized operations on these arrays.

🡺Numpy ndarray

🡪All of the numeric functionality of numpy is orchestrated by two important constituents of the numpy package, ndarray and Ufuncs (Universal function). Numpy ndarray is a multi-dimensional array object which is the core data container for all of the numpy operations. Universal functions are the functions which operate on ndarrays in an element by element fashion.

🡪Arrays (or matrices) are one of the fundamental representations of data. Mostly an array will be of a single data type (homogeneous) and possibly multi-dimensional sometimes. The numpy ndarray is a generalization of the same. Let’s get started with the introduction by creating an array.

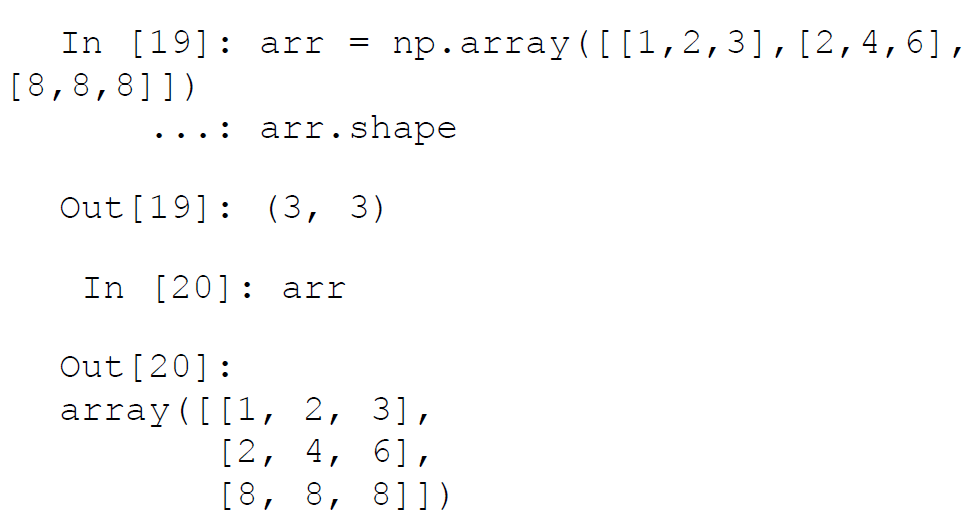


🡪In the previous example, we created a one-dimensional array from a normal list containing integers. The shape attribute of the array object will tell us about the dimensions of the array.

🡪One important thing to keep in mind is that all the elements in an array must have the same data type. If you try to initialize an array in which the elements are mixed, i.e. you mix some strings with the numbers then all of the elements will get converted into a string type and we won’t be able to perform most of the numpy operations on that array.

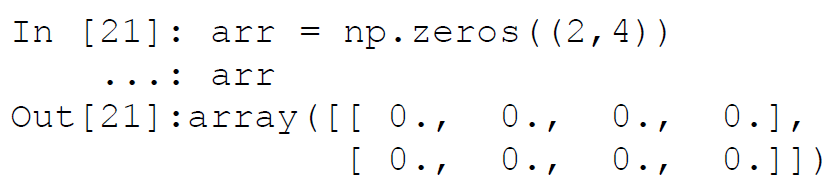
🡺Creating Arrays

🡪Arrays can be created in multiple ways in numpy. One of the ways was demonstrated earlier to create a single-dimensional array. Similarly we can stack up multiple lists to create a multidimensional array.

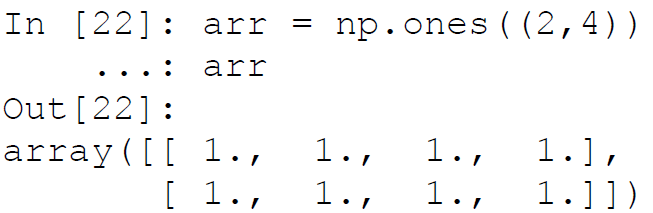


🡪In addition to this we can create arrays using a bunch of special functions provided by numpy.

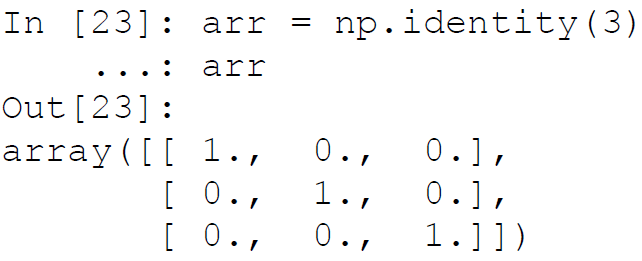
1. **np.zeros** : Creates a matrix of specified dimensions containing only zeroes:



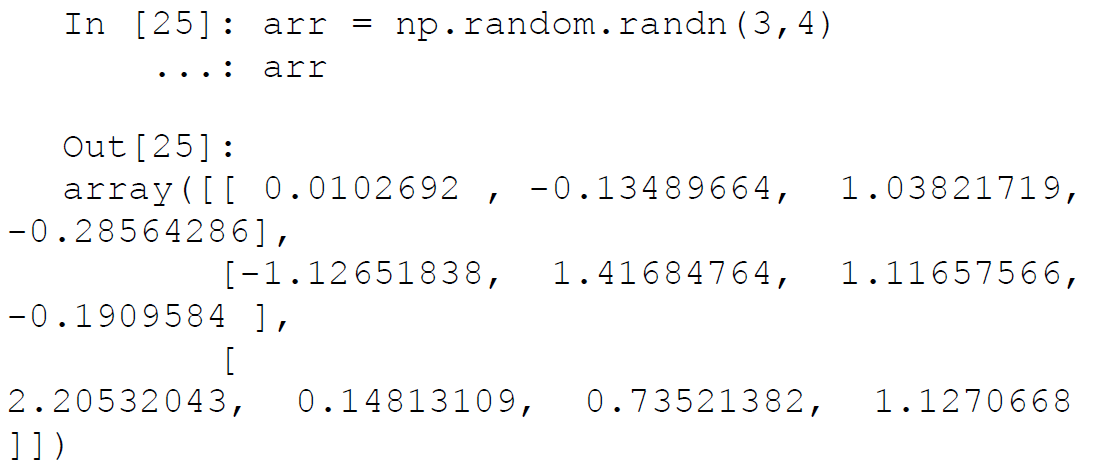
1. np.ones : Creates a matrix of specified dimension containing only ones:



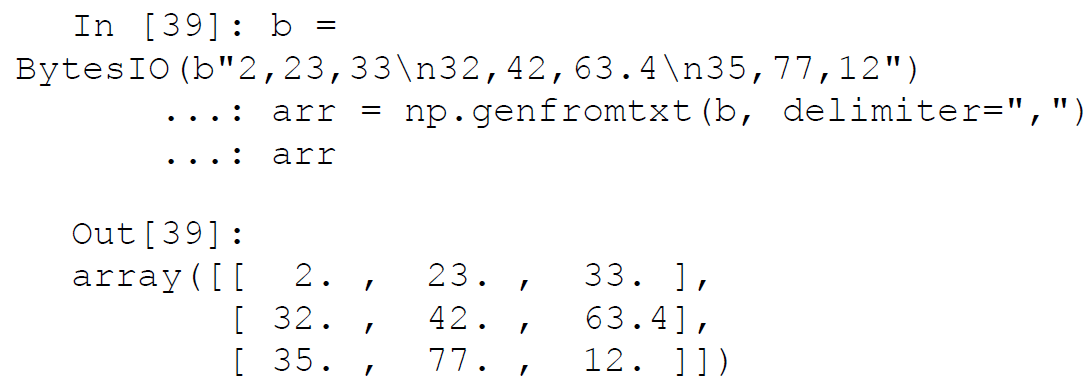
1. np.identity : Creates an identity matrix of specified dimensions:



🡪Often, an important requirement is to initialize an array of a specified dimension with random values. This can be done easily by using the randn function from the numpy.random package:



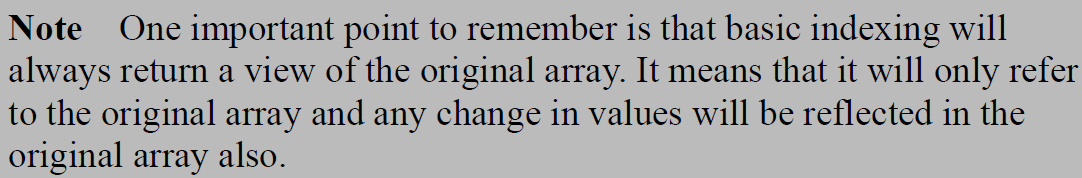
🡪One of the functions that we can use to read data from text file to a numpy array is genfromtext. This function can open a text file and read in data delimited by any character. (delimiter for a comma separated file is “,”). Since it is not our preferred way of retrieving data, we will give a brief example of the function here.



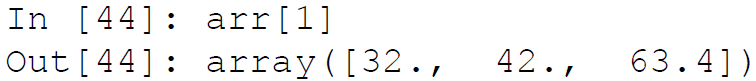
**🡺Accessing Array Elements**

🡪 Once we have created an array by reading in our data, the next important part is to access that data using a wide variety of mechanisms. Numpy provides a lot of ways in which array elements can be accessed. We will try to give the most popular useful ways that facilitate this.

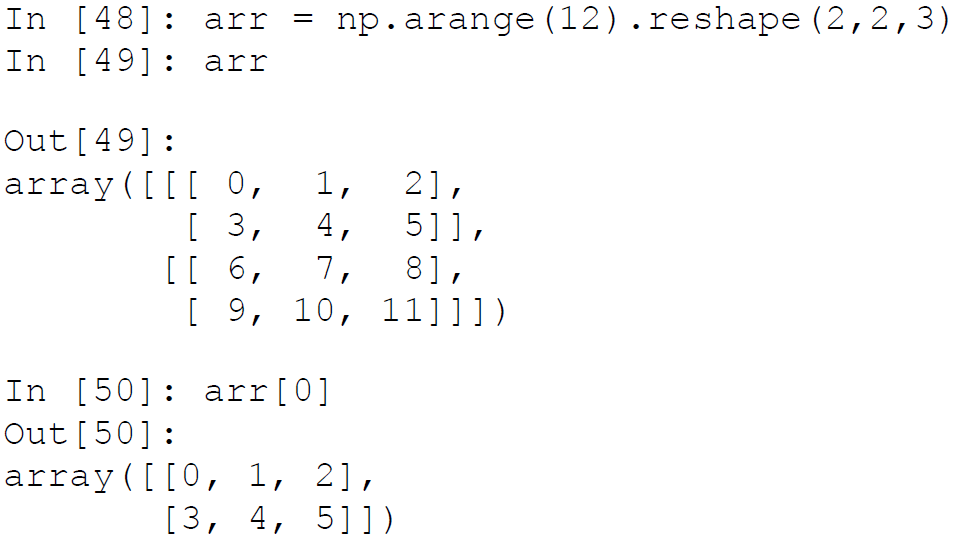
🡺Basic Indexing And Slicing : Ndarray can leverage the basic indexing operations that are followed by the list class, i.e. list object [obj]. If the obj is not an ndarray object, then the indexing is said to be basic indexing.



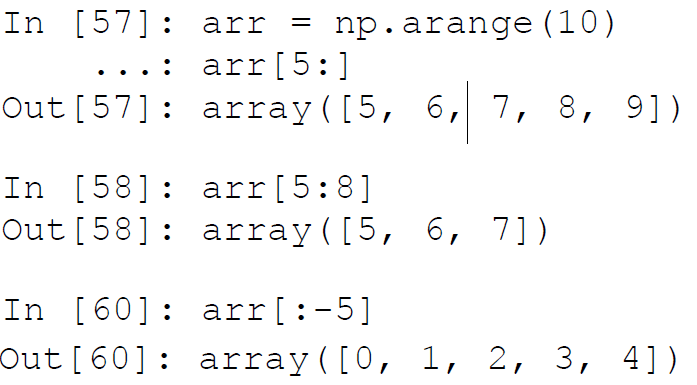
🡪 For example, if we want to access the complete second row of the array in one of the earlier examples, we can simply refer to it using arr[1] .



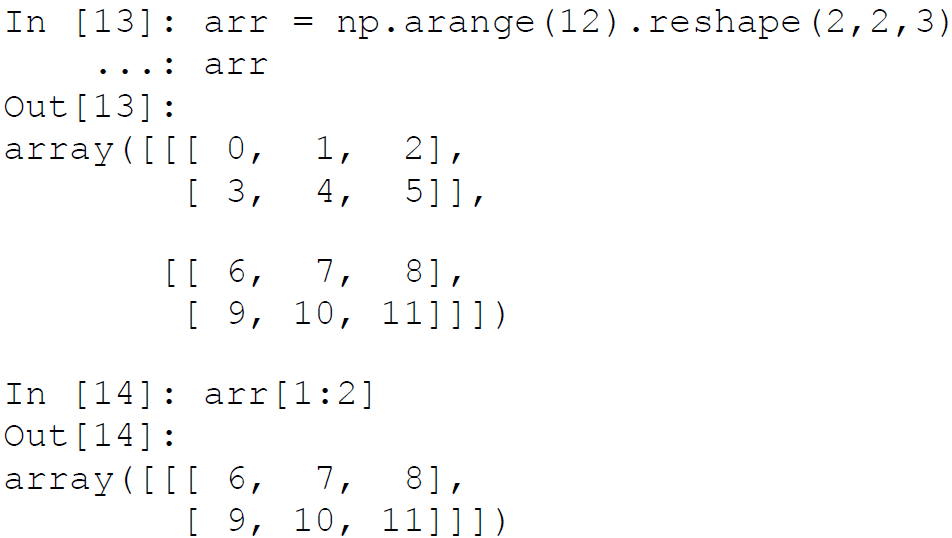
🡪 This access becomes interesting in the case of an array having more than two dimensions. Consider the following code snippet.



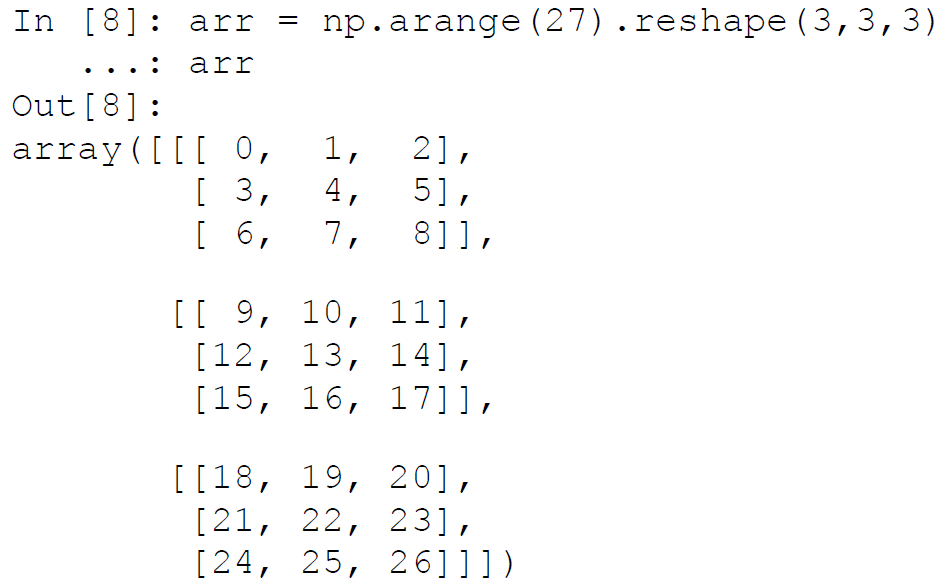
🡪 Here we see that using a similar indexing scheme as above, we get an array having one lesser dimension than the original array. The next important concept in accessing arrays is the concept of slicing arrays. Suppose we want to have a collection of elements only instead of all the elements. Then we can use slicing to access the elements. We will demonstrate the concept with a one-dimensional array.



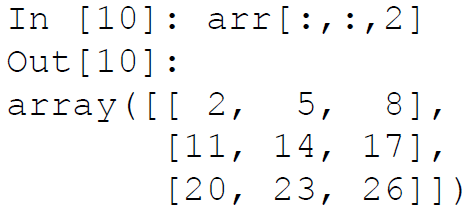
🡪 If the number of dimensions in the object supplied is less than the dimension of the array being accessed then the colon (:) is assumed for all the dimensions. Consider the following example



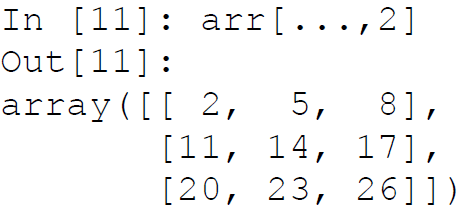
🡪 Another way to access an array is to use dots (…) based indexing. Suppose in a three-dimensional array we want to access the value of only one column. We can do it in two ways.



🡪 Now if we want to access the third column, we can use two different notations to access that column:

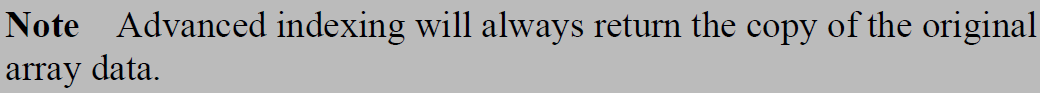


🡪 We can also use a dot notation in the following way. Both of the methods gets us the same value but the dot notation is concise. The dot notation stands for as many colons as required to complete an indexing operation.

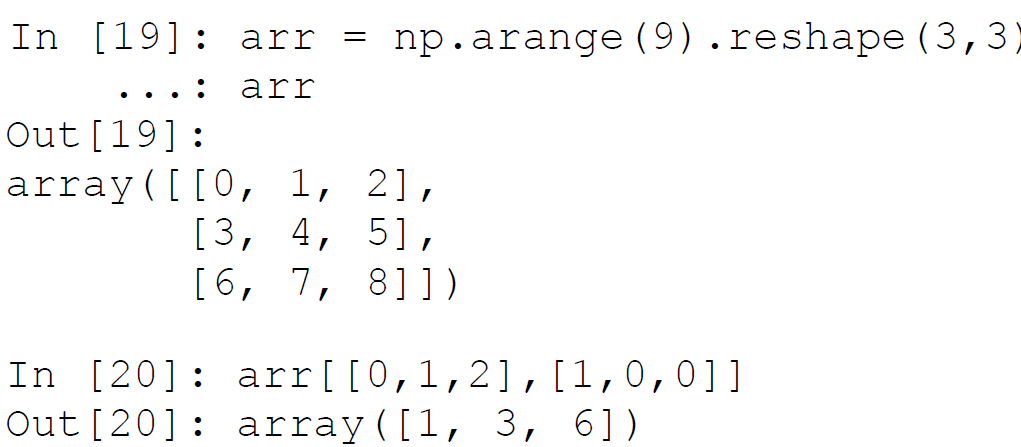


**🡺Advanced Indexing**

🡪 The difference in advanced indexing and basic indexing comes from the type of object being used to reference the array. If the object is an ndarray object (data type int or bool) or a non-tuple sequence object or a tuple object containing an ndarray (data type integer or bool), then the indexing being done on the array is said to be advanced indexing.

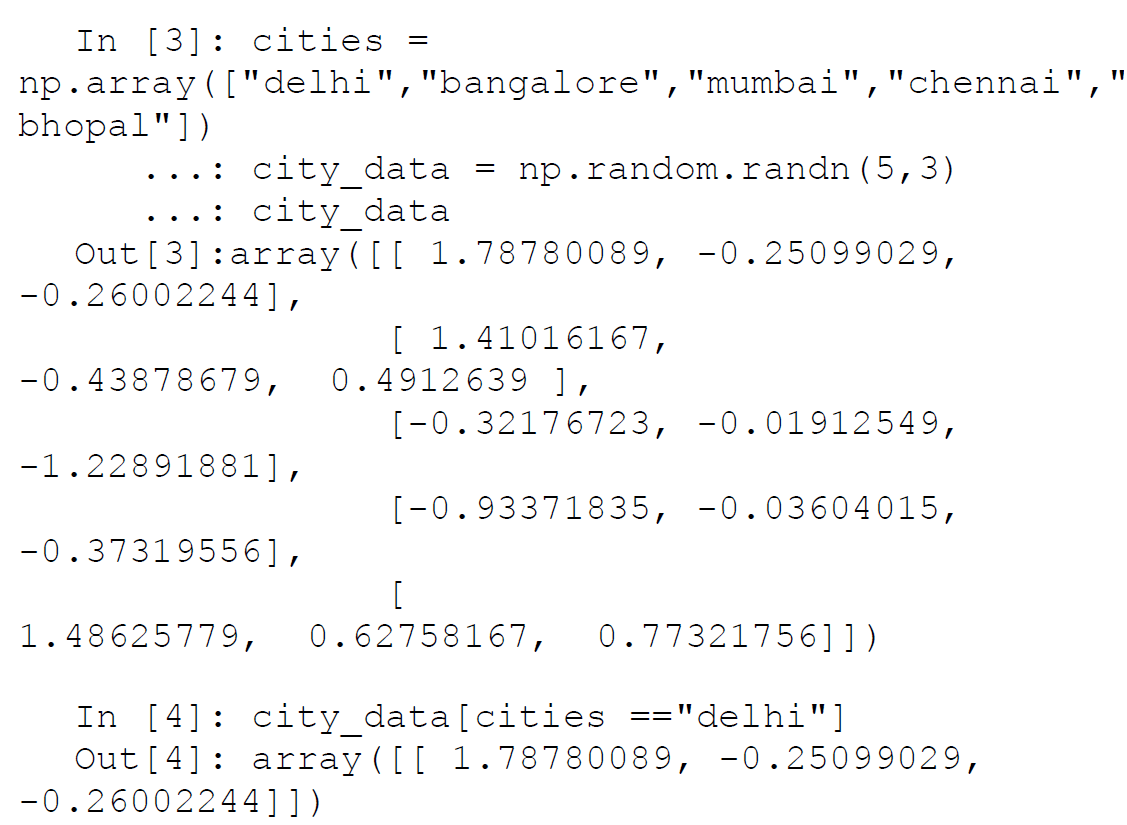


🡺**Integer array indexing** : This advanced indexing occurs when the reference object is also an array. The simplest type of indexing is when we provide an array that’s equal in dimensions to the array being accessed. For example:

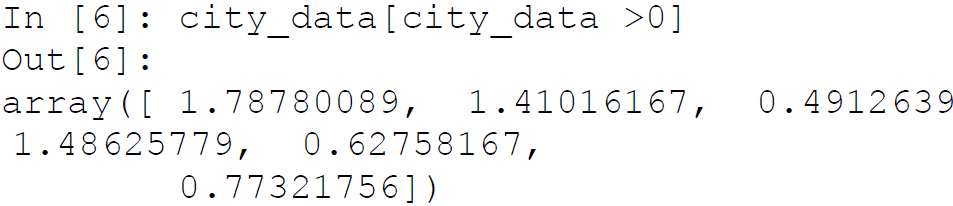


**🡪 In this example we have provided an array in which the first part identifies the rows we want to access and the second identifies the columns which we want to address. This is quite similar to providing a collective element-wise address.**

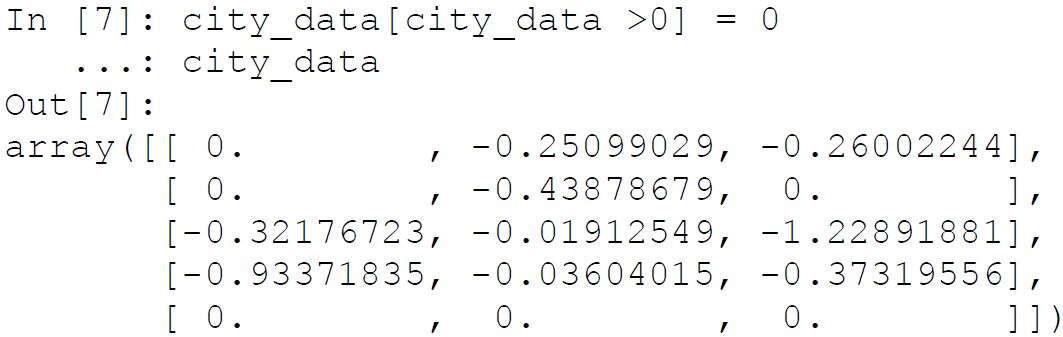
🡺**Boolean indexing** : This advanced indexing occurs when the reference object is an array of Boolean values. This is used when we want to access data based on some conditions, in that case, Boolean indexing can be used. We will illustrate it with an example. Suppose in one array, we have the names of some cities and in another array, we have some data related to those cities.



🡪 We can also use Boolean indexing for selecting some elements of an array that satisfy a particular condition. For example, in the previous array suppose we want to only select non-zero elements. We can do that easily using the following code.



🡪 We observe that the shape of the array is not maintained so we directly cannot always use this indexing method. But this method is quite useful in doing conditional data substitution. Suppose in the previous case, we want to substitute all the non-zero values with 0. We can achieve that operation by the following code.

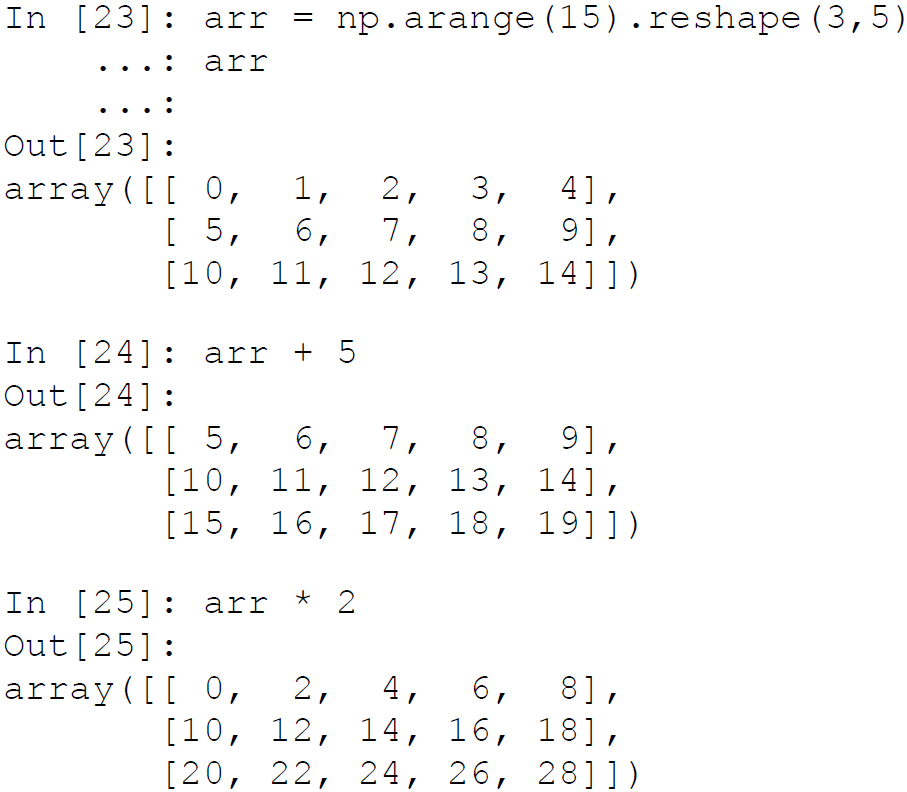


**🡺Operations on Arrays**

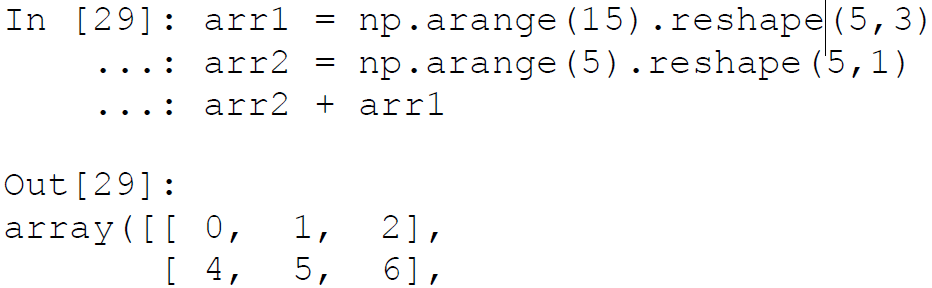
🡪 At the start of this section, we mentioned the concept of Universal functions (Ufuncs). In this sub-section, we learn some of the functionalities provided by those functions. Most of the operations on the numpy arrays is achieved by using these functions. Numpy provides a rich set of functions that we can leverage for various operations on arrays. We cover some of those functions in brief, but we recommend you to always refer to the official documentation of the project to learn more and leverage them in your own projects.

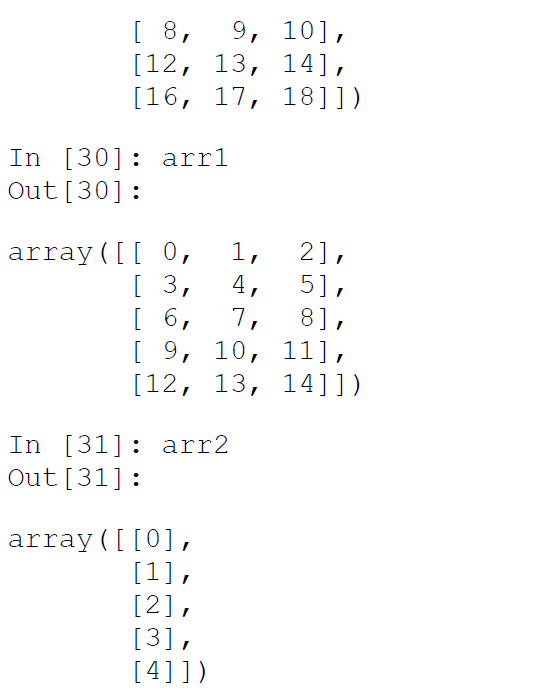
🡪Universal functions are functions that operate on arrays in an element by element fashion. The implementation of Ufunc is vectorized, which means that the execution of Ufuncs on arrays is quite fast. The Ufuncs implemented in the numpy package are implemented in compiled C code for speed and efficiency. But it is possible to write custom functions by extending the numpy.ufunc class of the numpy package.

🡪Ufuncs are simple and easy to understand once you are able to relate the output they produce on a particular array.

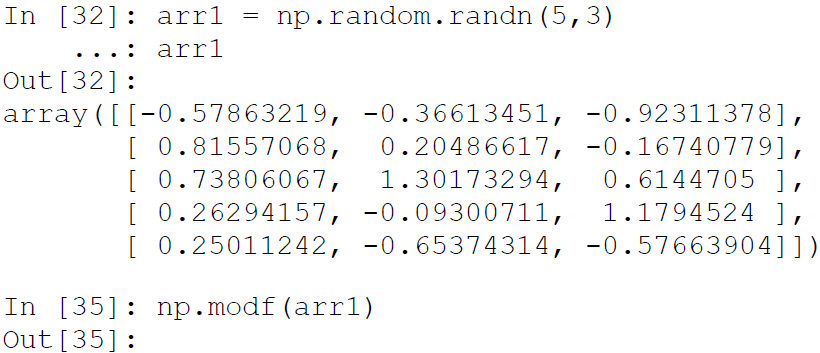


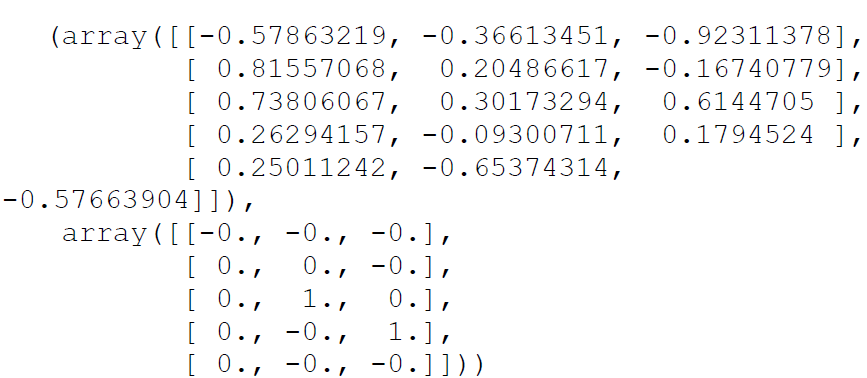
🡪 We see that the standard operators when used in conjunction with arrays work element-wise. Some Ufuncs will take two arrays as input and output a single array, while a rare few will output two arrays also.





🡪 Here we see that we were able to add up two arrays even when they were of different sizes. This is achieved by the concept of broadcasting . We will conclude this brief discussion on operations on arrays by demonstrating a function that will return two arrays.



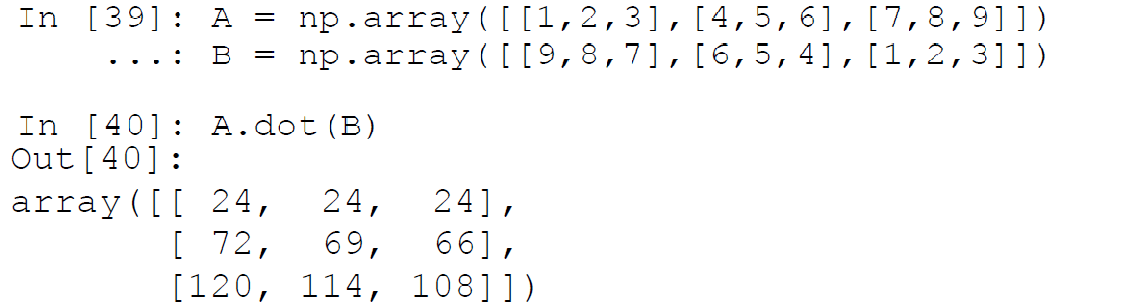


🡪 The function modf will return the fractional and the integer part of the input supplied to it. Hence it will return two arrays of the same size.

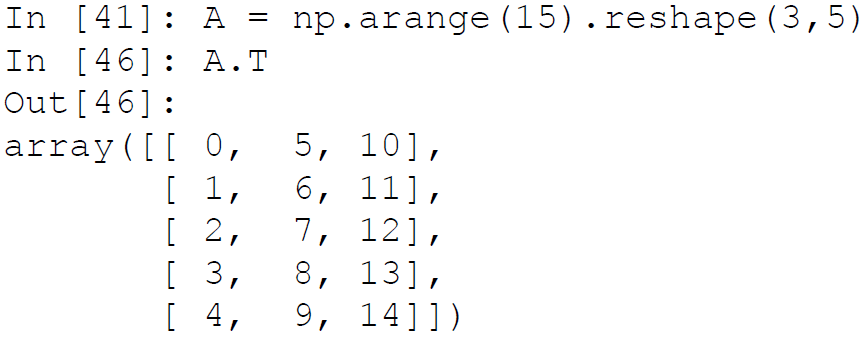
**🡺Linear Algebra Using numpy**

🡪 Linear algebra is an integral part of the domain of Machine Learning. Most of the algorithms we will deal with can be concisely expressed using the operations of linear algebra. Numpy was initially built to provide the functions similar to MATLAB and hence linear algebra functions on arrays were always an important part of it. In this section, we learn a bit about performing linear algebra on ndarrays using the functions implemented in the numpy package.

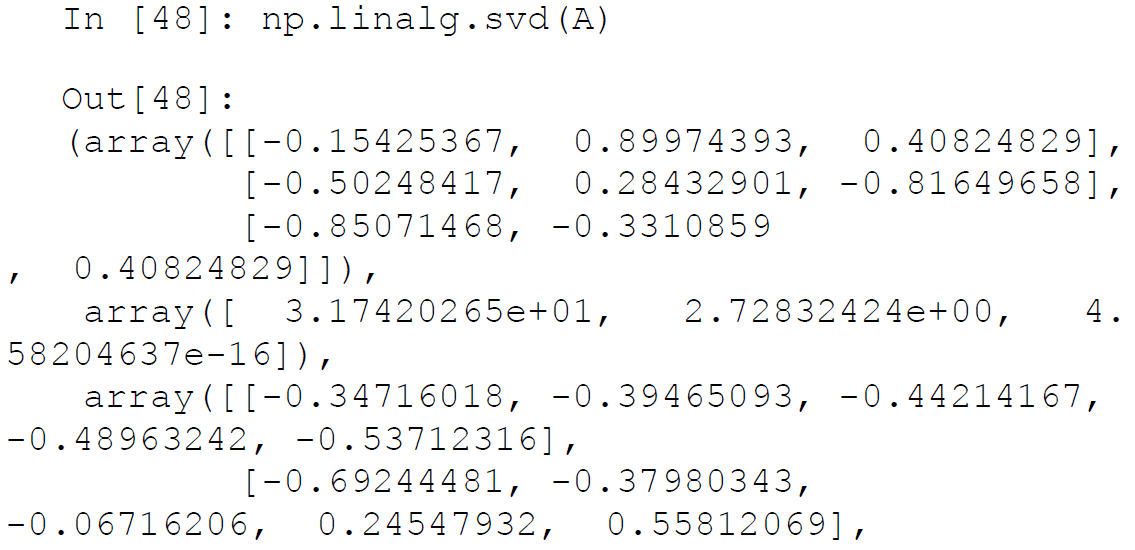
🡪 One of the most widely used operations in linear algebra is the dot product. This can be performed on two compatible (brush up on your matrices and array skills if you need to know which arrays are compatible for a dot product) ndarrays by using the dot function.

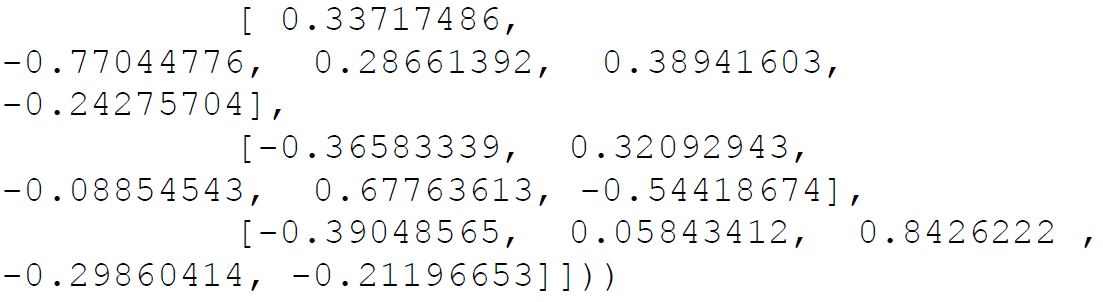


🡪 Similarly, there are functions implemented for finding different products of matrices like inner, outer, and so on. Another popular matrix operation is transpose of a matrix. This can be easily achieved by using the T function.

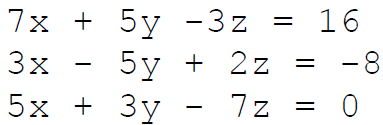


🡪 Oftentimes, we need to find out decomposition of a matrix into its constituents factors. This is called matrix factorization. This can be achieved by the appropriate functions. A popular matrix factorization method is SVD factorization (covered briefly in Chapter 1 concepts), which returns decomposition of a matrix into three different matrices. This can be done using linalg.svd function.

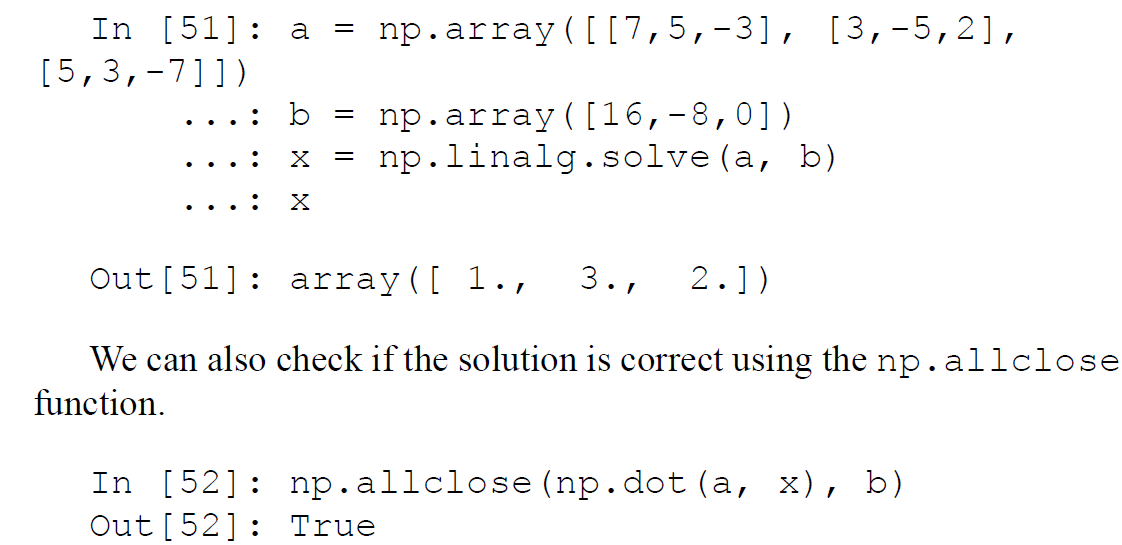




🡪 Linear algebra is often also used to solve a system of equations. Using the matrix notation of system of equations and the provided function of numpy, we can easily solve such a system of equation. Consider the system of equations:



🡪 This can be represented as two matrices : the coefficient matrix (a in the example) and the constants vector (b in the example).



🡪 Similarly, functions are there for finding the inverse of a matrix, eigen vectors and eigen values of a matrix, norm of a matrix, determinant of a matrix, and so on

**🡺Pandas**

🡪 Pandas is an important Python library for data manipulation, wrangling, and analysis. It functions as an intuitive and easy-to-use set of tools for performing operations on any kind of data. Initial work for pandas was done by Wes McKinney in 2008 while he was a developer at AQR Capital Management.

🡺Data Structures Of Pandas

🡪All the data representation in pandas is done using two primary data structures:

1. Series
2. Dataframes

🡺Series

🡪 Series in pandas is a one-dimensional ndarray with an axis label. It means that in functionality, it is almost similar to a simple array. The values in a series will have an index that needs to be hashable. This requirement is needed when we perform manipulation and summarization on data contained in a series data structure. Series objects can be used to represent time series data also. In this case, the index is a datetime object.

🡺Dataframe

🡪 Dataframe is the most important and useful data structure, which is used for almost all kind of data representation and manipulation in pandas. Unlike numpy arrays (in general) a dataframe can contain heterogeneous data.

🡪Typically tabular data is represented using dataframes, which is analogous to an Excel sheet or a SQL table. This is extremely useful in representing raw datasets as well as processed feature sets in Machine Learning and Data Science. All the operations can be performed along the axes, rows, and columns, in a dataframe. This will be the primary data structure which we will leverage, in most of the use cases in our later chapters.

🡺Data Retrieval

🡪 Pandas provides numerous ways to retrieve and read in data. We can convert data from CSV files, databases, flat files, and so on into dataframes. We can also convert a list of dictionaries (Python dict) into a dataframe. The sources of data which pandas allows us to handle cover almost all the major data sources. For our introduction, we will cover three of the most important data sources:

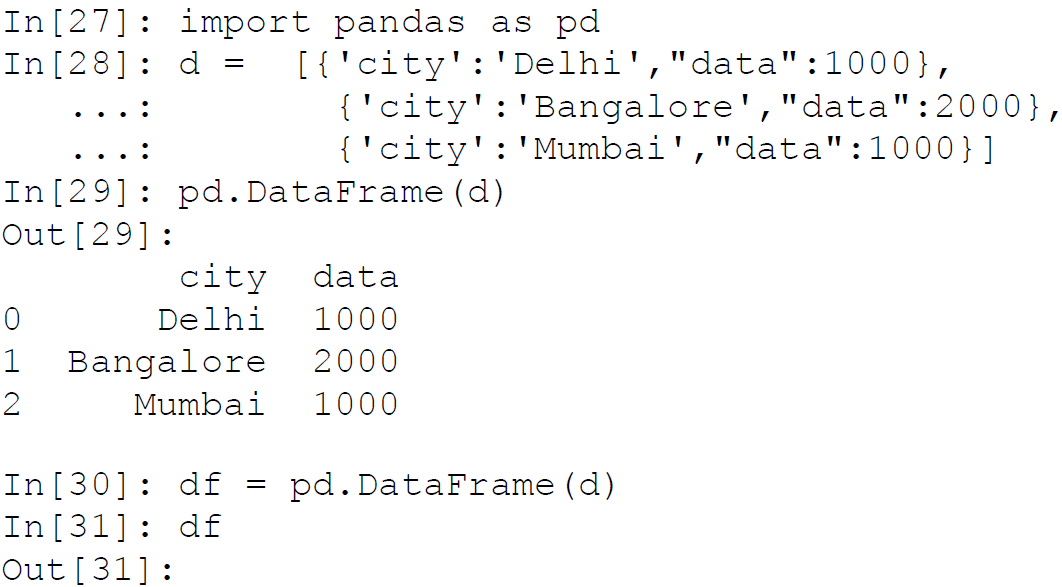
List of dictionaries

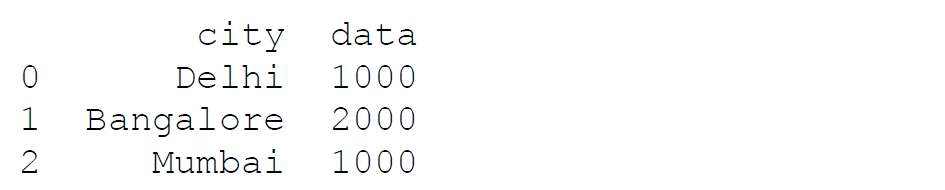
CSV files

Databases

🡺List Of Dictionaries to Dataframe

🡪 This is one of the simplest methods to create a dataframe . It is useful in scenarios where we arrive at the data we want to analyze, after performing some computations and manipulations on the raw data. This allows us to integrate a pandas based analysis into data being generated by other Python processing pipelines.





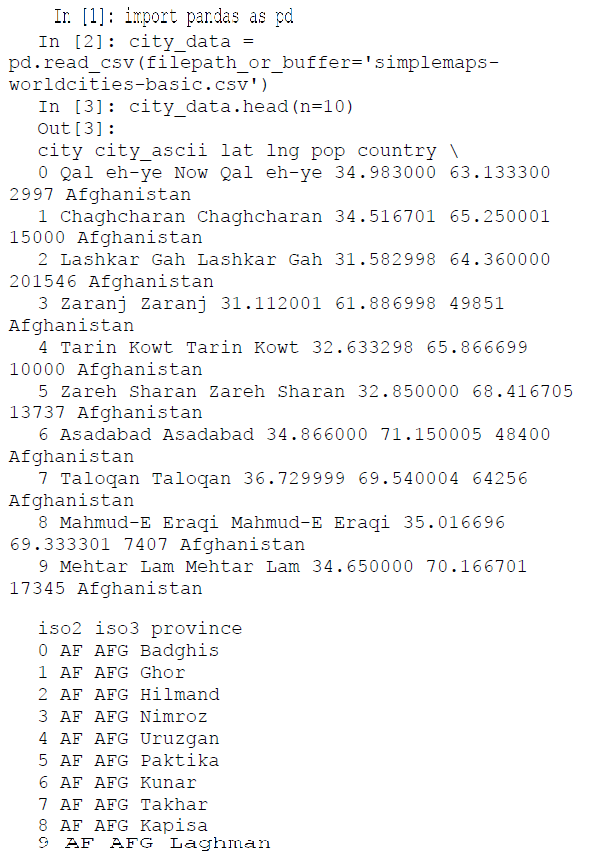
🡪 Here we provided a list of Python dictionaries to the DataFrame class of the pandas library and the dictionary was converted into a DataFrame. Two important things to note here: first the keys of dictionary are picked up as the column names in the dataframe (we can also supply some other name as arguments for different column names), secondly we didn’t supply an index and hence it picked up the default index of normal arrays.

🡺CSV Files to Dataframe

🡪 CSV (Comma Separated Files) files are perhaps one of the most widely used ways of creating a dataframe. We can easily read in a CSV, or any delimited file (like TSV), using pandas and convert into a dataframe. For our example we will read in the following file and convert into a dataframe by using Python. The data in Figure below is a sample slice of a CSV file containing the data of cities of the world from http://simplemaps.com/data/world-cities .



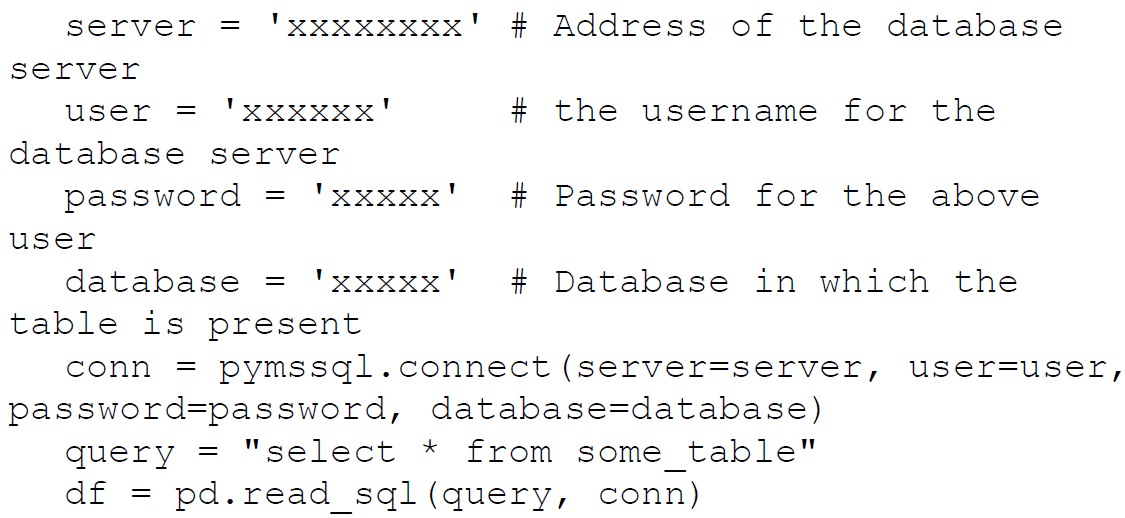
🡪 We can convert this file into a dataframe with the help of the following code leveraging pandas.



🡪 As the file we supplied had a header included, those values were used as the name of the columns in the resultant dataframe. This is a very basic yet core usage of the function pandas.read\_csv. The function comes with a multitude of parameters that can be used to modify its behavior as required. We will not cover the entire gamut of parameters available and you are encouraged to read the documentation of this function as this is one of the starting point of most Python based data analysis.

**🡺Database to Dataframe**

🡪 The most important data source for data scientists is the existing data sources used by their organizations. Relational databases (DBs) and data warehouses are the de facto standard of data storage in almost all of the organizations. Pandas provides capabilities to connect to these databases directly, execute queries on them to extract data, and then convert the result of the query into a structured dataframe. The pandas.from\_sql function combined with Python’s powerful database library implies that the task of getting data from DBs is simple and easy. Due to this capability, no intermediate steps of data extraction are required. We will now take an example of reading data from a Microsoft SQL Server database. The following code will achieve this task.



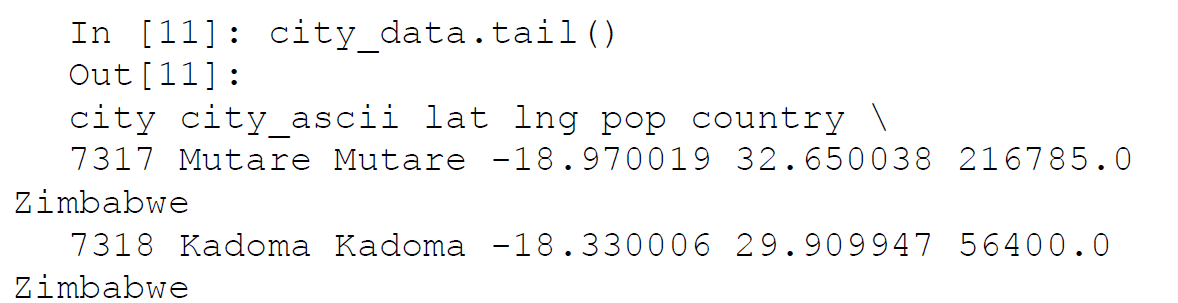
🡪 The important to thing to notice here is the connection object (conn in the code). This object is the one which will identify the database server information and the type of database to pandas. Based on the endpoint database server we will change the connection object**. For example we are using the pymssql library for access to Microsoft SQL server here**. If our data source is changed to a Postgres database, the **connection object will change but the rest of the procedure will be similar**. This facility is really handy when we need to perform similar analyses on data originating from different sources. Once again, the read\_sql function of pandas provides a lot of parameters that allow us to control its behavior. We also recommend you to check out the sqlalchemy library, which makes creating connection objects easier irrespective of the type of database vendor and also provides a lot of other utilities.

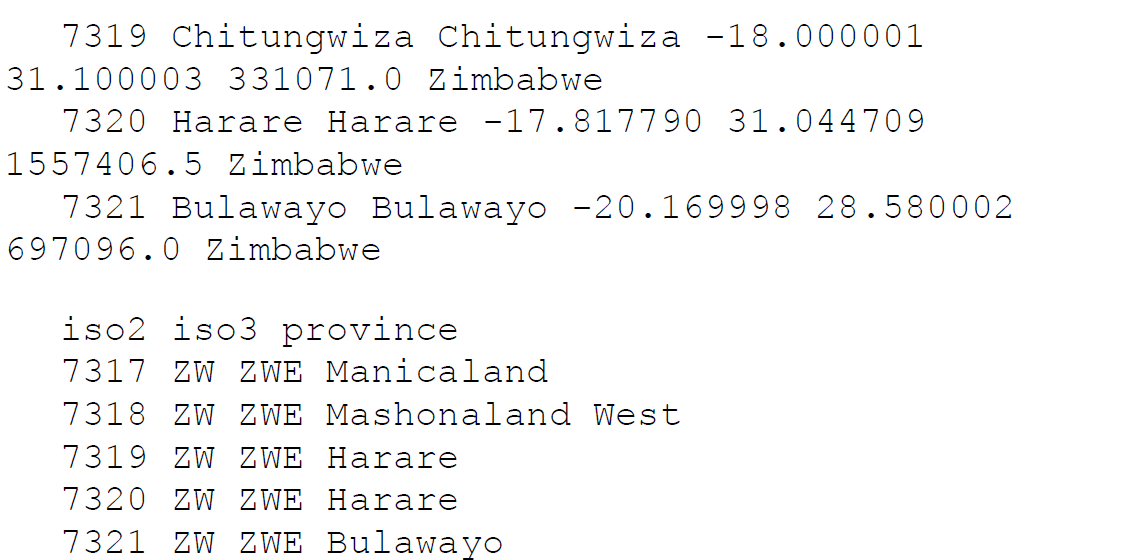
1. Data Access

🡪 The most important part after reading in our data is that of accessing that data using the data structure’s access mechanisms. Accessing data in the pandas dataframe and series objects is very much similar to the access mechanism that exist for Python lists or numpy arrays. But they also offer some extra methods for data access specific to data frame/series.

**🡺Head and Tail**

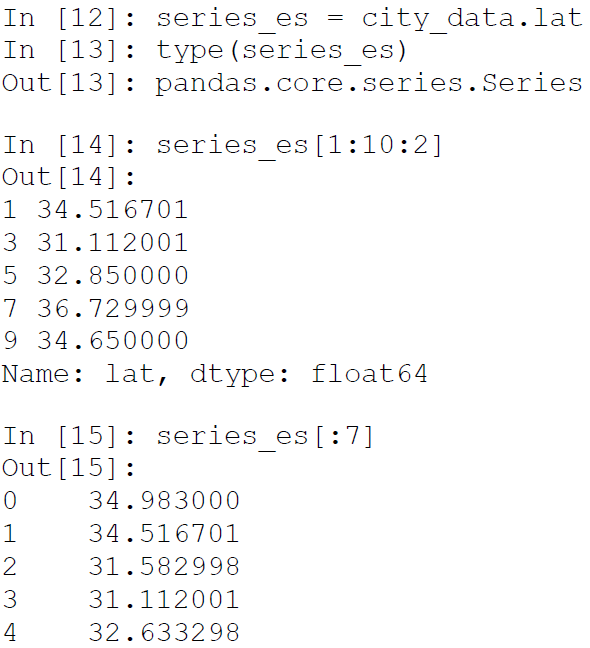
🡪 In the previous section we witnessed the method head. It gives us the first few rows (by default 5) of the data. A corresponding function is tail, which gives us the last few rows of the dataframe. These are one of the most widely used pandas functions, as we often need to take a peek at our data as and when we apply different operations/selections on it. We already have seen the output of head, so we’ll use the tail function on the same dataframe and see its output.





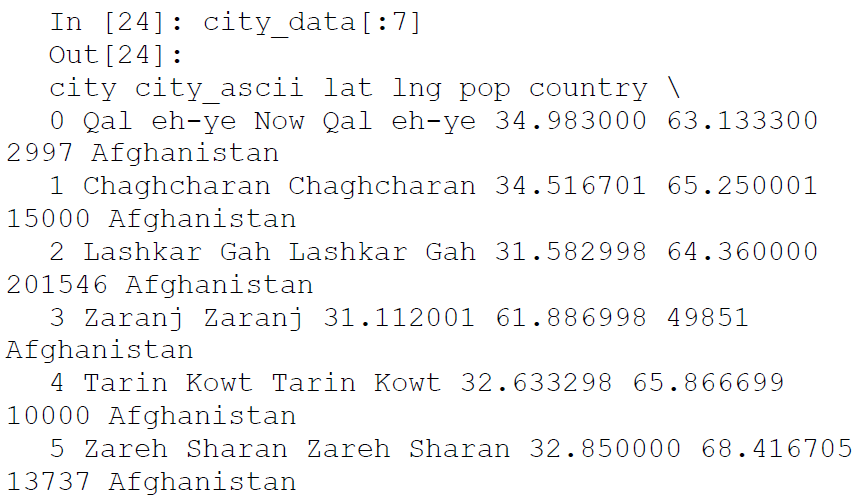
**🡺Slicing and Dicing**

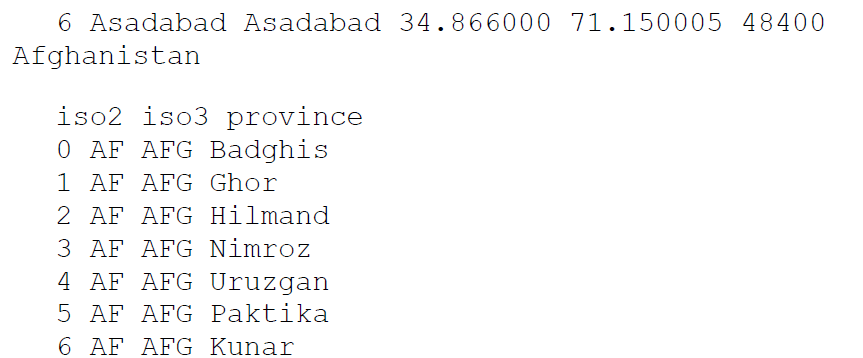
🡪 The usual rules of slicing and dicing data that we used in Python lists apply to the Series object as well.



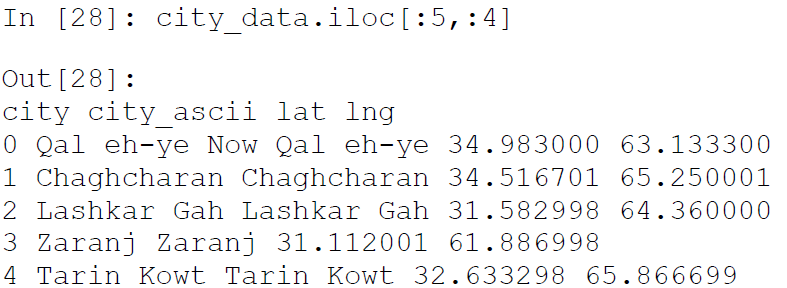


🡪 The examples given here are self-explanatory and you can refer to the numpy section for more details. Similar slicing rules apply for dataframes also but the only difference is that now simple slicing refers to the slicing of rows and all the other columns will end up in the result. Consider the following example

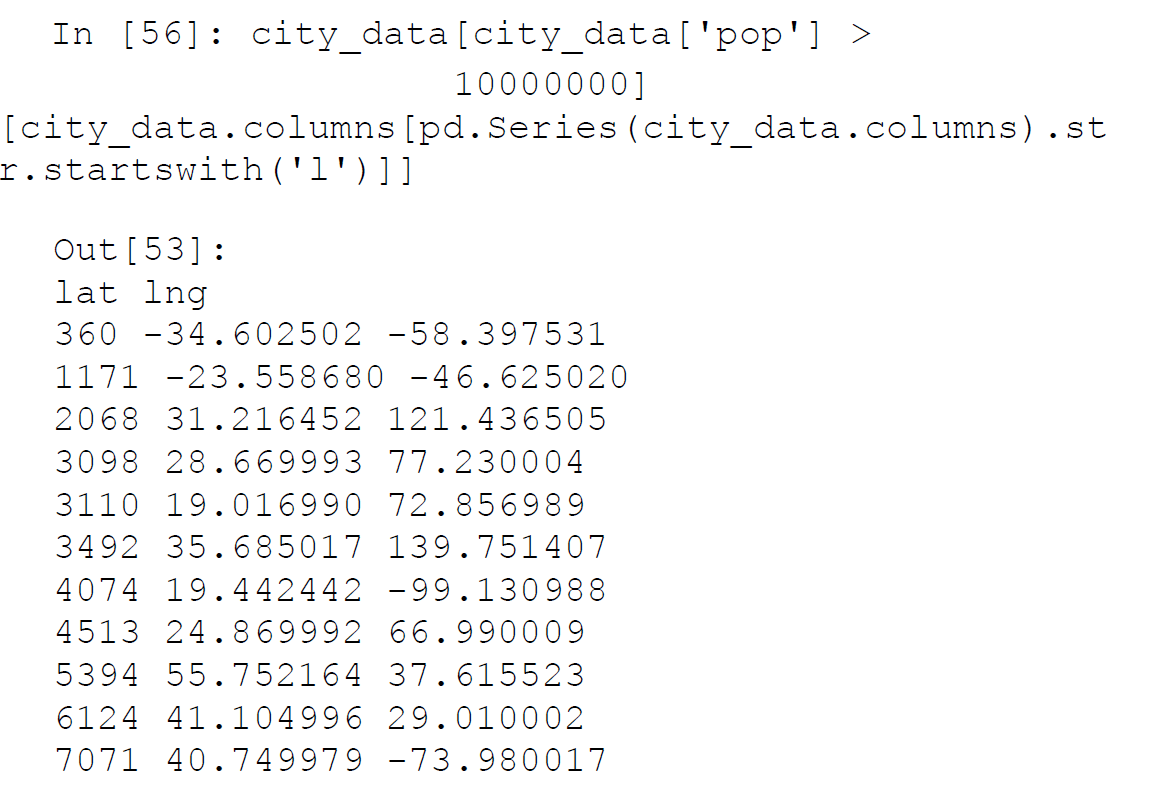




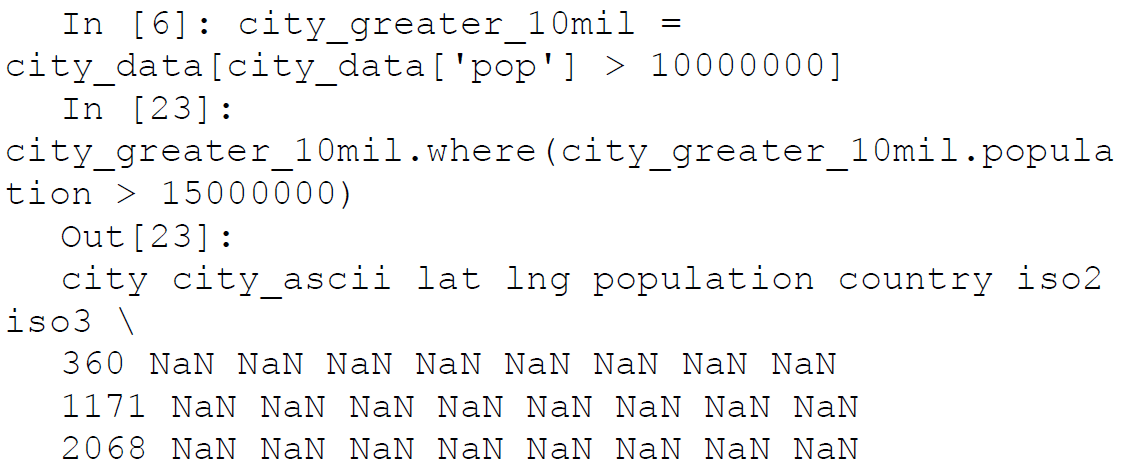
🡪 For providing access to specific rows and specific columns, pandas provides useful functions like iloc and loc which can be used to refer to specific rows and columns in a dataframe. There is also the ix function but we recommend using either loc or iloc. The following examples leverages the iloc function provided by pandas. This allows us to select the rows and columns using structure similar to array slicing. In the example, we will only pick up the first five rows and the first four columns.

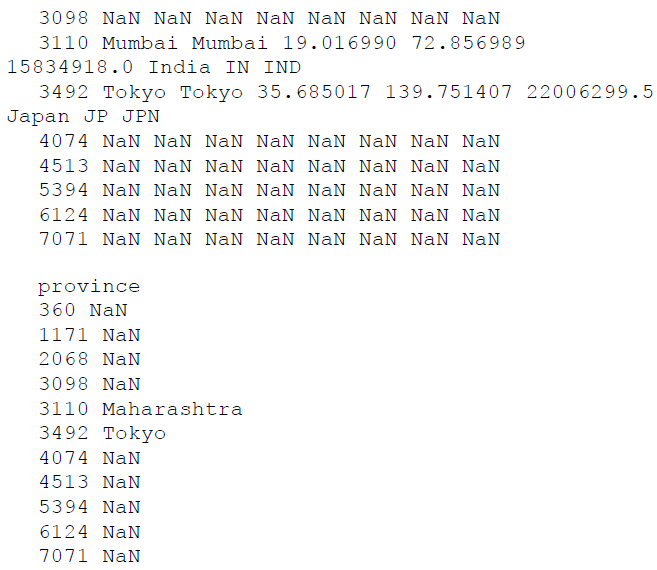


🡪 Another access mechanism is Boolean based access to the dataframe rows or columns. This is particularly important for dataframes, as it allows us to work with a specific set of rows and columns. Let’s consider the following example in which we want to select cities that have population of more than 10 million and select columns that start with the letter l:



🡪 When we select data based on some condition, we always get the part of dataframe that satisfies the condition supplied. **Sometimes we want to test a condition against a dataframe but want to preserve the shape of the dataframe**. In these cases, we can use the where function (check out numpy's where function also to see the analogy!). **We’ll illustrate this function with an example in which we will try to select all the cities that have population greater than 15 million.**





🡪 Here we see that we get the output dataframe of the same size but the rows that don’t conform to the condition are replaced with NaN.

🡪In this section, we learned some of the core data access mechanisms of pandas dataframes. The data access mechanism of pandas are as simple and extensive to use as with numpy this ensures that we have various way to access our data.

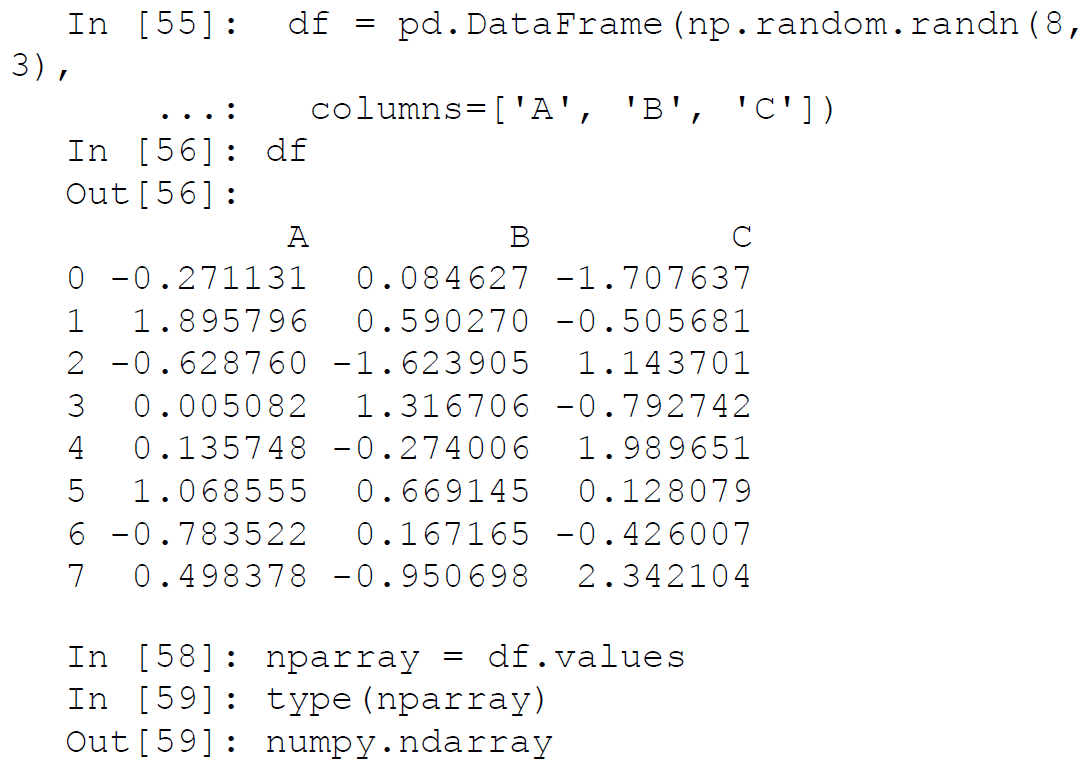
1. Data Operations

🡪 In subsequent chapters of our book, the pandas dataframe will be our data structure of choice for most data processing and wrangling operations. So we would like to spend some more time exploring some important operations that can be performed on dataframes using specific supplied functions.

🡺Values Attribute

🡪 Each pandas dataframe will have certain attributes. One of the important attributes is values. It is important as it allows us access to the raw values stored in the dataframe and if they all homogenous i.e., of the same kind then we can use numpy operations on them. This becomes important when our data is a mix of numeric and other data types and after some selections

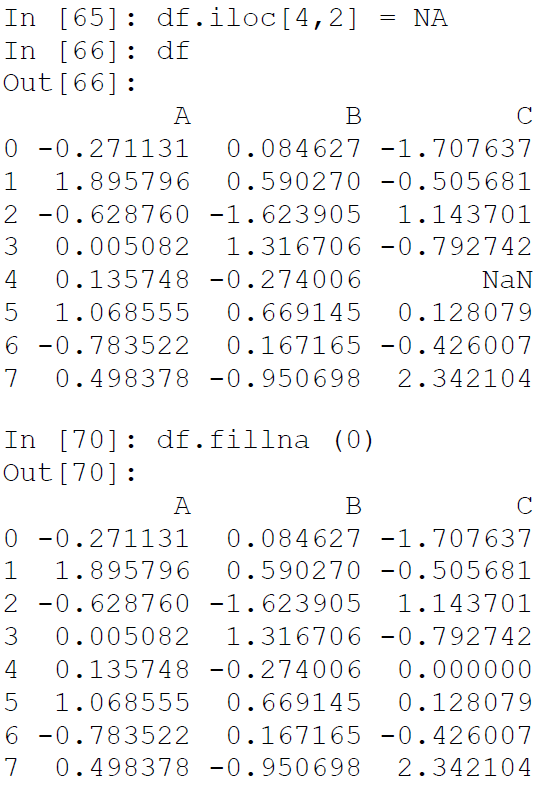
and computations, we arrive at the required subset of numeric data. Using the values attribute of the output dataframe, we can treat it in the same way as a numpy array. This is very useful when working with feature sets in Machine Learning. Traditionally, numpy vectorized operations are much faster than function based operations on dataframes.



🡺Missing Data and the fillna Function

🡪 In real-world datasets, the data is seldom clean and polished. We usually will have a lot of issues with data quality (missing values, wrong values and so on). One of the most common data quality issues is that of missing data. Pandas provides us with a convenient function that allows us to handle the missing values of a dataframe.

🡪 For demonstrating the use of the fillna function , we will use the dataframe we created in the previous example and introduce missing values in it.

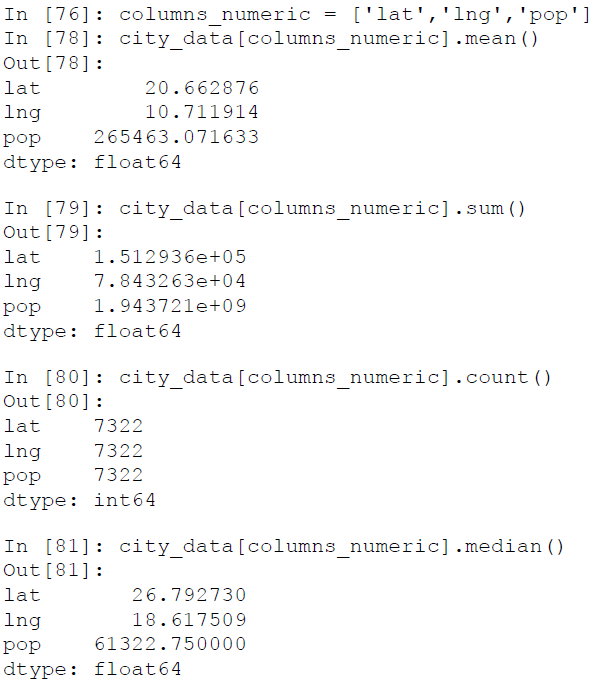


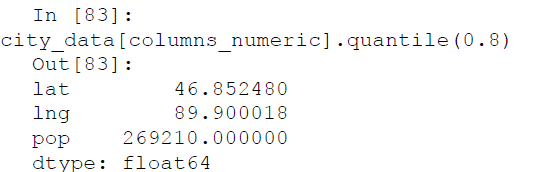
🡪 Here we have substituted the missing value with a default value. We can use a variety of methods to arrive at the substituting value (mean, median, and so on). We will see more methods of missing value treatment (like imputation) in subsequent chapters.

🡺Descriptive Statistics Functions

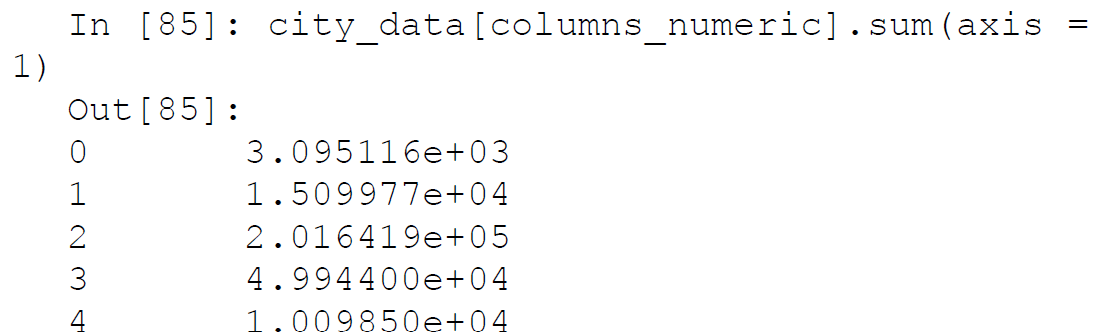
🡪 A general practice of dealing with datasets is to know as much about them as possible. Descriptive statistics of a dataframe give data scientists a comprehensive look into important information about any attributes and features in the dataset. Pandas packs a bunch of functions, which facilitate easy access to these statistics.

🡪 Consider the cities dataframe (city\_data) that we consulted in the earlier section. We will use pandas functions to gather some descriptive statistical information about the attributes of that dataframe. As we only have three numeric columns in that particular dataframe, we will deal with a subset of the dataframe which contains only those three values.

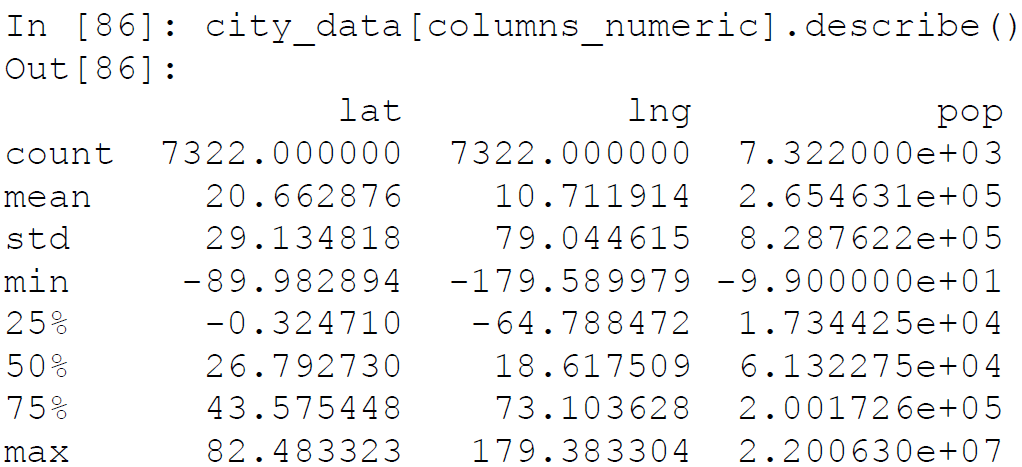




🡪 All these operations were applied to each of the columns, the default behavior. We can also get all these statistics for each row by using a different axis. This will give us the calculated statistics for each row in the dataframe.



🡪 Pandas also provides us with another very handy function called describe . This function will calculate the most important statistics for numerical data in one go so that we don’t have to use individual functions.



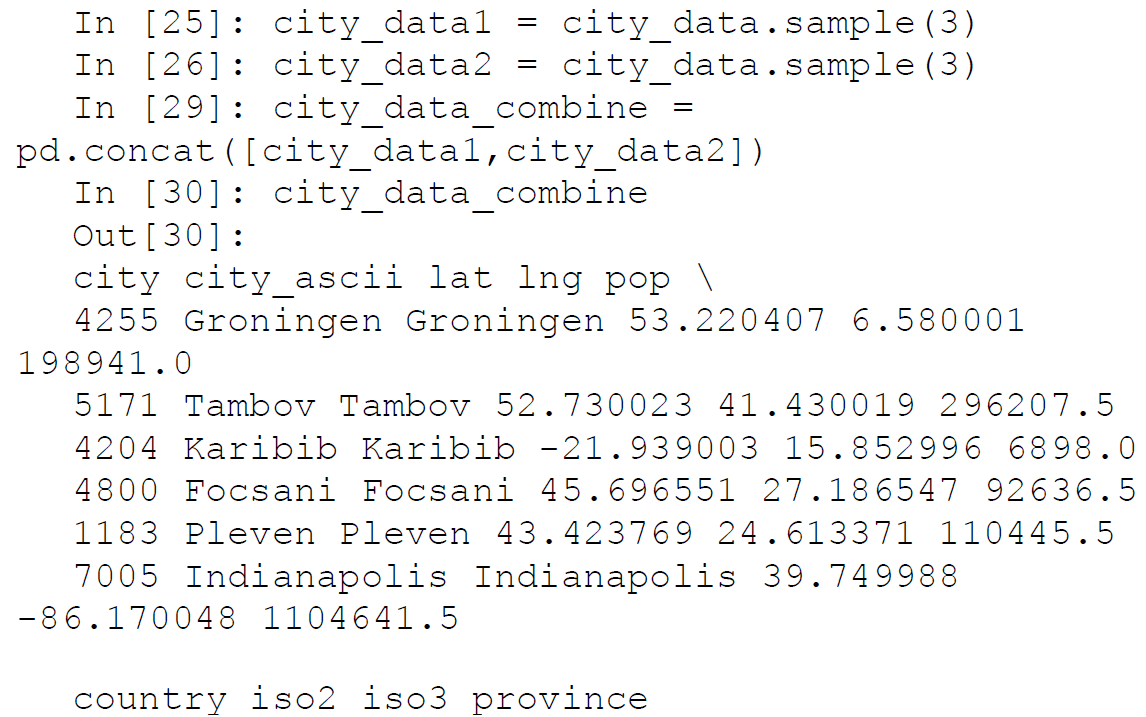
1. Concatenating Dataframes

🡪 Most Data Science projects will have data from more than one data source. These data sources will mostly have data that’s related in some way to each other and the subsequent steps in data analysis will require them to be concatenated or joined. Pandas provides a rich set of functions that allow us to merge different data sources. We cover a small subset of such methods. In this section, we explore and learn about two methods that can be used to perform all kinds of amalgamations of dataframes.

**🡺Concatenating Using the concat method**

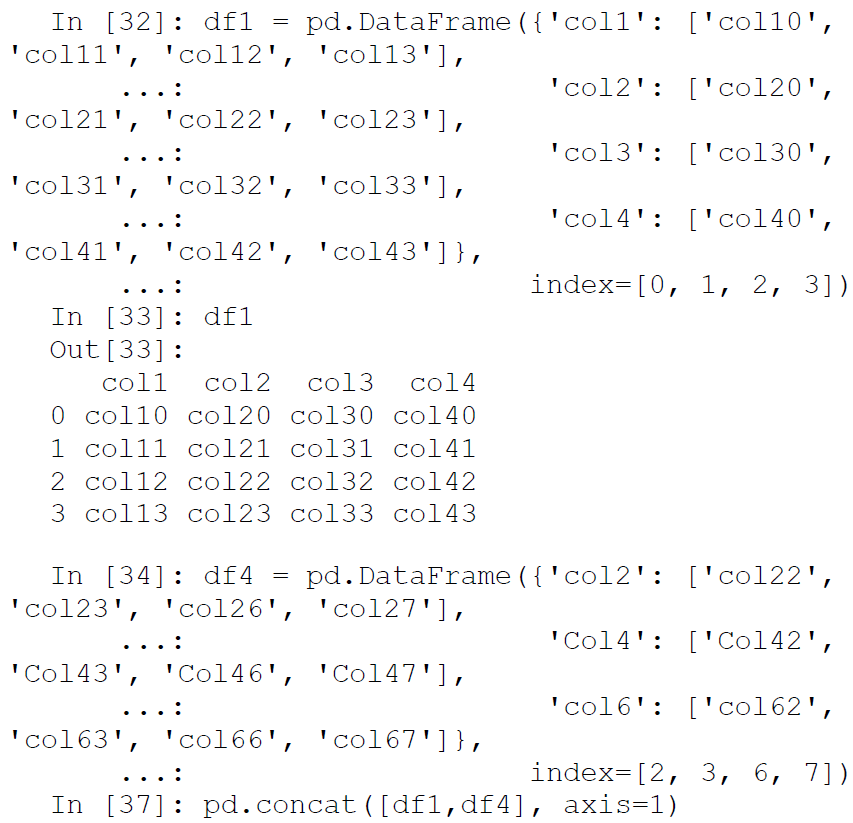
🡪 The first method to concatenate different dataframes in pandas is by using the concat method. The majority of the concatenation operations on dataframes will be possible by tweaking the parameters of the concat method. Let’s look at a couple of examples to understand how the concat method works.

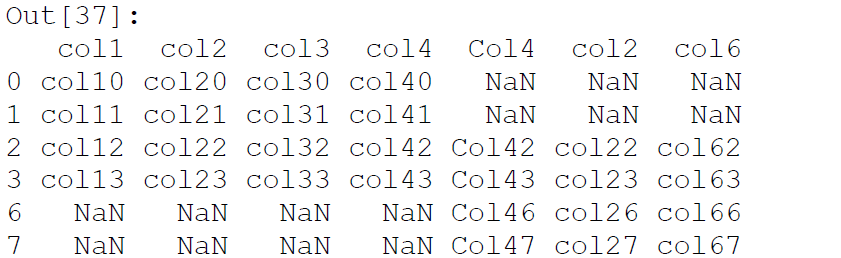
🡪The simplest scenario of concatenating is when we have more than one fragment of the same dataframe (which may happen if you are reading it from a stream or in chunks). In that case, we can just supply the constituent dataframes to the concat function as follows.





🡪 Another common scenario of concatenating is when we have information about the columns of same dataframe split across different dataframes. Then we can use the concat method again to combine all the dataframes. Consider the following example.



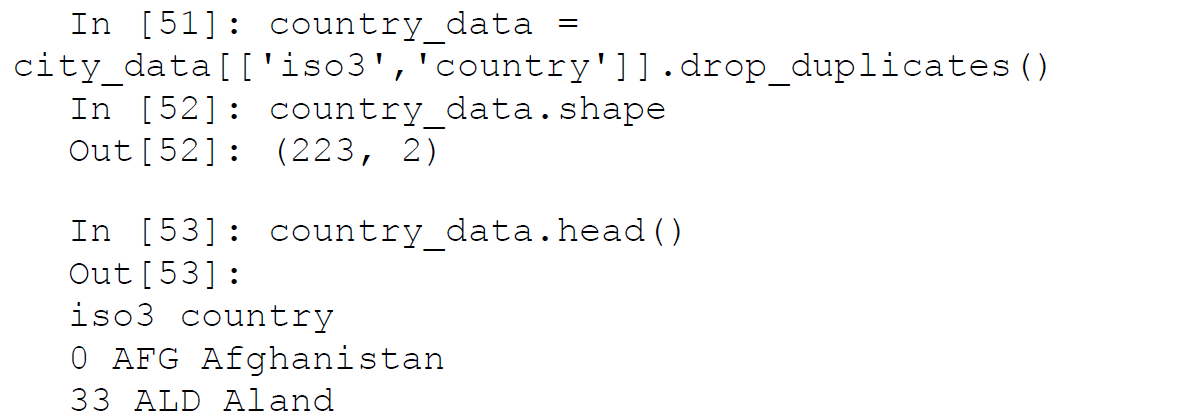


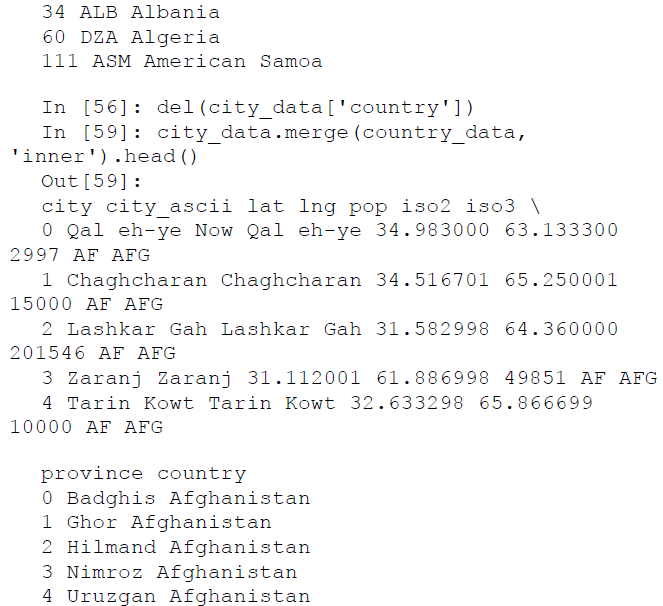
🡺Database Style Concatenations Using the merge Command

🡪 The most familiar way to concatenate data (for those acquainted with relational databases) is using the join operation provided by the databases. Pandas provides a database friendly set of join operations for dataframes. These operations are optimized for high performance and are often the preferred method for joining disparate dataframes.

🡪**Joining by columns** : This is the most natural way of joining two dataframes. In this method, we have two dataframes sharing a common column and we can join the two dataframes using that column. The pandas library has a full range of join operations (inner, outer, left, right, etc.) and we will demonstrate the use of inner join in this sub-section. You can easily figure out how to do the rest of join operations by checking out the pandas documentation.

🡪 For this example, we will break our original cities data into two different dataframes, one having the city information and the other having the country information. Then, we can join them using one of the shared common columns.





🡪 Here we had a common column in both the dataframes, iso3, which the merge function was able to pick up automatically. In case of the absence of such common names, we can provide the column names to join on, by using the parameter on of the merge function. The merge function provides a rich set of parameters that can be used to change its behavior as and when required. We will leave it on you to discover more about the merge function by trying out a few examples.

1. Scikit-learn

🡪 Scikit-learn is one of the most important and indispensable Python frameworks for Data Science and Machine Learning in Python. **It implements a wide range of Machine Learning algorithms covering major areas of Machine Learning like classification, clustering, regression, and so on**.

🡪All the mainstream Machine Learning algorithms like support vector machines, logistic regression, random forests, K-means clustering, hierarchical clustering, and many many more, are implemented efficiently in this library. Perhaps this library forms the foundation of applied and practical Machine Learning. Besides this, its easy-to-use API and code design patterns have been widely adopted across other frameworks too!

🡪 **The scikit-learn project was initiated as a Google summer of code project by David Cournapeau**.

🡪 Scikit-learn is mostly written in Python but for providing a better performance some of the core code is written in **Cython**. It also uses wrappers around popular implementations of learning algorithms like logistic regression (using LIBLINEAR) and support vector machine (using LIBSVM).

🡪 In our introduction of scikit-learn we will first go through the

basic design principles of the library and then build on this theoretical knowledge of the package. We will implement some of the algorithms on sample data to get you acquainted with the basic syntax. We leverage scikit-learn extensively in subsequent chapters, so the intent here is to acquaint you with how the library is structured and its core components.

🡺Core APIs

This framework is built on quite a small and simple list of core API ideas and design patterns. In this section we will briefly touch on the core APIs on which the central operations of scikitlearn are based.

1. **Dataset representation** : The data representation of most Machine Learning tasks are quite similar to each other. Very often we will have a collection of data points represented by a stacking of data point vectors. Basically considering a dataset, each row in the dataset represents a vector for a specific data point observation.

**🡪 A data point vector contains multiple independent variables (or features) and one or more dependent variables (response variables).**

🡪 For example, if we have a linear regression problem which can be represented as [(X 1 , X2 , X 3 , X 4 , …, X n ), (Y)] where the independent variables (features) are represented by the Xs and the dependent variable (response variable) is represented by Y. The idea is to predict Y by fitting a model on the features This data representation resembles a matrix (considering multiple data point vectors), and a natural way to depict it is by using numpy arrays.

🡪 This choice of data representation is quite simple yet powerful as we are able to access the powerful functionalities and the efficient nature of vectorized numpy array operations.

🡪 **In fact recent updates of scikit-learn even accept pandas dataframes as inputs instead of explicitly needing you to convert them to feature arrays!**

1. **Estimators** : The estimator interface is one of the most important components of the scikit-learn library. **All the Machine Learning algorithms in the package implement the estimator interface**. **The learning process is handled in a two-step process.**

🡪 **The first step is the initialization of the estimator object; this involves selecting the appropriate class object for the algorithm and supplying the parameters or hyperparameters for it.**

**🡪 The second step is applying the fit function to the data supplied (feature set and response variables). The fit function will learn the output parameters of the Machine Learning algorithm and expose them as public attributes of the object for easy inspection of the final model.**

🡪 The data to the fit function is generally supplied in the form of an input-output matrix pair. In addition to the Machine Learning algorithms, several data transformation mechanisms are also implemented using the estimators APIs (for example, scaling of features, PCA, etc.). This allows for simple data transformation and a simple mechanism to expose transformation mechanisms in a consistent way.

1. **Predictors** : The predictor interface is implemented to generate predictions, forecasts, etc. using a learned estimator for unknown data. For example, in the case of a supervised learning problem, the predictor interface will provide predicted classes for the unknown test array supplied to it.

🡪 Predictor interface also contains support for providing quantified values of the output it supplies. A requirement of a predictor implementation is to provide a score function; this function will provide a scalar value for the test input provided to it which will quantify the effectiveness of the model used. Such values will be used in the future for tuning our Machine Learning models.

1. **Transformers** : Transformation of input data before learning of a model is a very common task in Machine Learning. Some data transformations are simple, for example replacing some missing data with a constant, taking a log transform, while some data transformations are similar to learning algorithms themselves (for example, PCA).

🡪 To simplify the task of such transformations, some estimator objects will implement the transformer interface. This interface allows us to perform a non-trivial transformation on the input data and supply the output to our actual learning algorithm.

🡪 Since the transformer object will retain the estimator used for transformation, it becomes very easy to apply the same transformation to unknown test data using the transform function.

**🡺Advanced APIs**

🡪 In the earlier section we saw some of the most basic tenets of the scikitlearn package. In this section we will briefly touch on the advanced constructs that are built on those basics. These advanced set of APIs will often help data scientists in expressing a complex set of essential operations using a simple and stream-lined syntax.

1. **Meta estimators** : The meta estimator interface (implemented using the multiclass interface) is a collection of estimators which can be composed by accumulating simple binary classifiers. It allows us to extend the binary classifiers to implement multi-class, multi-label, multi-regression, and multi-class-multi-label classifications. This interface is important as these scenarios are common in modern day Machine Learning and the capability to implement this out-of-the-box reduces the programming requirements for data scientists. We should also remember that most binary estimators in the scikit-learn library have multiclass capabilities built in and we won’t be using the meta-estimators unless we need custom behavior.
2. **Pipeline and features unions** : The steps of Machine Learning are mostly sequential in nature. We will read in the data, apply some simple or complex transformations, fit an appropriate model, and predict using the model for unseen data. Another hallmark of the Machine Learning process is the iteration of these steps multiple times due to its iterative nature, to arrive at the best possible model and then deploy the same.

🡪It is convenient to chain these operations together and repeat them as a single unit instead of applying operations piecemeal. This concept is also known as Machine Learning pipelines. Scikit-learn provides a Pipeline API to achieve similar purpose.

🡪A Pipeline() object from the pipeline module can chain

multiple estimators together(transformations, modeling, etc.) and the resultant object can be used as an estimator itself.

🡪In addition to the pipeline API, which applies these estimators in a sequential method, we also have access to a FeatureUnion API, which will perform a specified set of operation in parallel and show the output of all the parallel operations.

🡪The use of pipelines is a fairly advanced topic and it will be made clearer, when we specifically see an example in the subsequent chapters.

1. **Model tuning and selection** : Each learning algorithm will have a bunch of parameters or hyperparameters associated with it. The iterative process of Machine Learning aims at finding the best set of parameters that give us the model having the best performance. For example, the process of tuning various hyperparameters of a random forest algorithm, to find the set which gives the best prediction accuracy (or any other performance metric).

🡪This process sometimes involves traversing through the parameter space, searching for the best parameter set. Do note that even though we mention the term parameter here, we typically indicate the hyperparameters of a model. Scikit-learn provides useful APIs that help us navigate this parameter space easily to find the best possible parameter combinations.

🡪We can use two meta-estimators—GridSearchCV

and RandomizedSearchCV—for facilitating the search of the best parameters. GridSearchCV, as the name suggests, involves providing a grid of possible parameters and trying each possible combination among them to arrive at the best one. An optimized approach often is to use a random search through the possible parameter set; this approach is provided by the RandomizedSearchCV API.

🡪It samples the parameters and avoids the combinatorial explosions that can result in the case of a higher number of parameters. In addition to the parameter search, these model selection methods also allow us to use different cross-validation schemes and score functions to measure performance.

1. Scikit-learn Example: Regression Models

🡪In the first chapter, we discussed an example which involved the task of classification. In this section, we will tackle another interesting Machine Learning problem, that of regression. Keep in mind the focus here is to introduce you to the basic steps involved in using some of the scikitlearn library APIs. We will not try to over-engineer our solution to arrive at the best model. Future chapters will focus on those aspects with real world datasets.

🡪For our regression example, we will use one of the datasets bundled

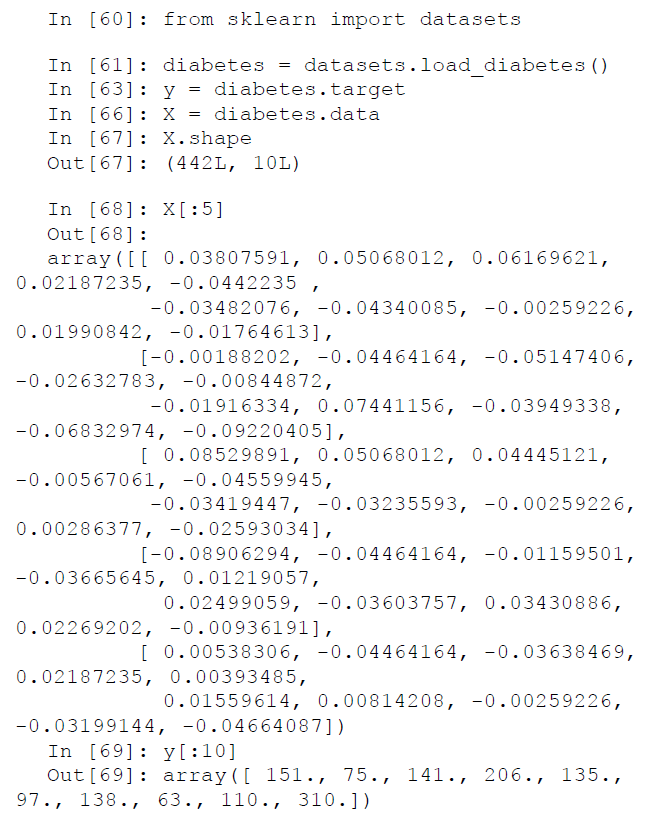
with the scikit-learn library, the diabetes dataset.

🡺The Dataset

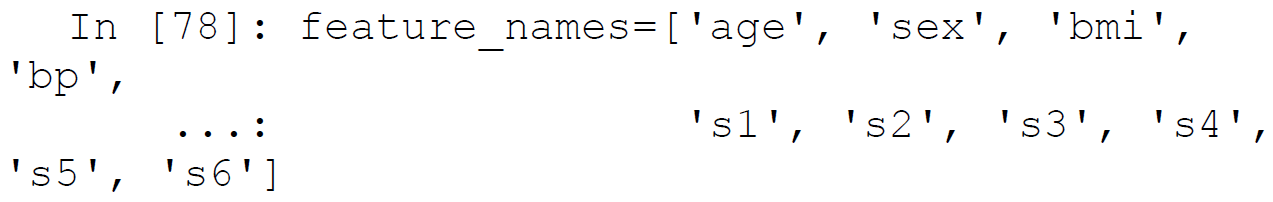
🡪The diabetes dataset is one of the bundled datasets with the scikitlearn library. This small dataset allows the new users of the library to learn and experiment various Machine Learning concepts, with a well known dataset. It contains observations of 10 baseline variables, age, sex, body mass index, average blood pressure. and six blood serum measurements for 442 diabetes patients. The dataset bundled with the package is already standardized (scaled), i.e. they have zero mean and unit L2 norm. The response (or target variable) is a quantitative measure of disease progression one year after baseline. The dataset can be used to answer two questions:

* + What is the baseline prediction of disease progression for future patients?
  + Which independent variables (features) are important factors for predicting disease -progression?

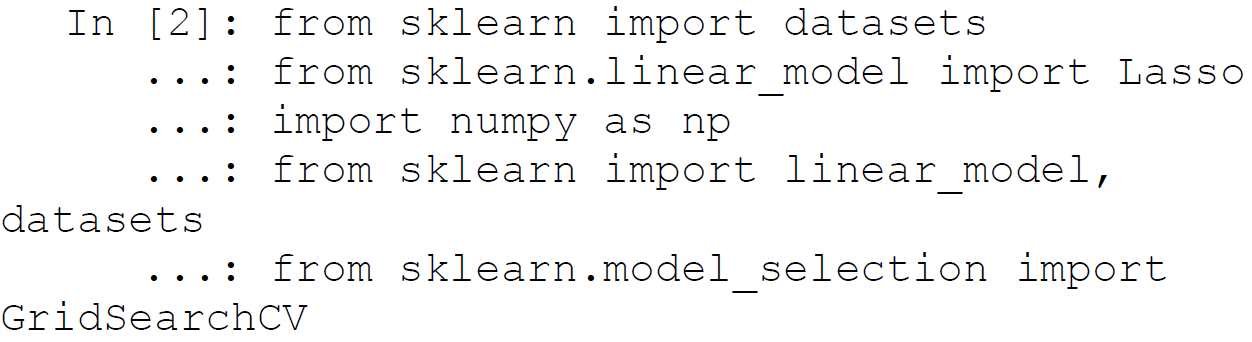
🡪We will try to answer the first question here by building a simple linear regression model. Lets get started by loading data



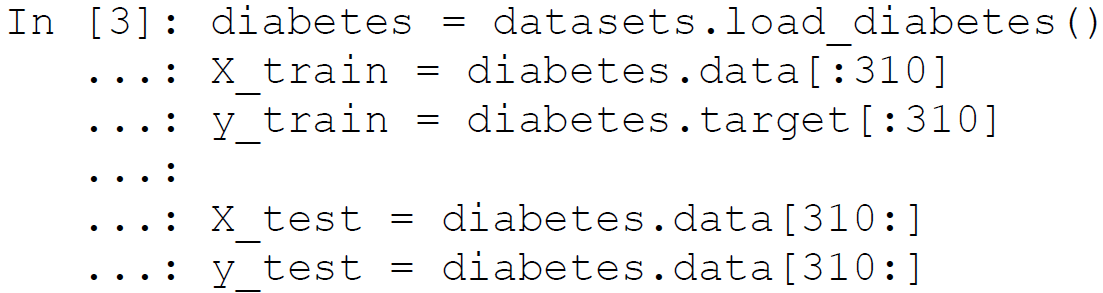
🡪Since we are using the data in the form of numpy arrays, we don’t get the name of the features in the data itself. But we will keep the reference to the variable names as they may be needed later in our process or just for future reference.



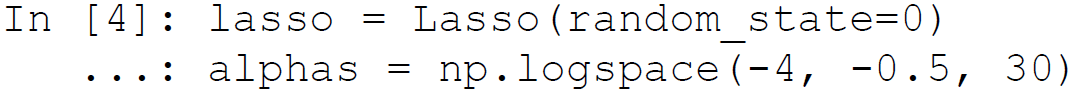
🡪For prediction of the response variable here, we will learn a Lasso model. A Lasso model is an extension of the normal linear regression model which allows us to apply L1 regularization to the model. Simply put, a lasso regression will try to minimize the number of independent variables in the final model. This will give us the model with the most important variables only (feature selection).



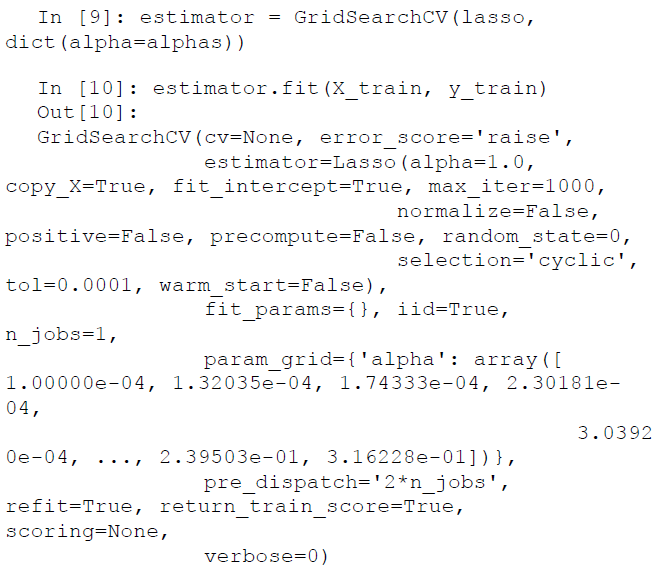
🡪We will split our data into separate test and train sets of data (train is used to train the model and test is used for model performance testing and evaluation).



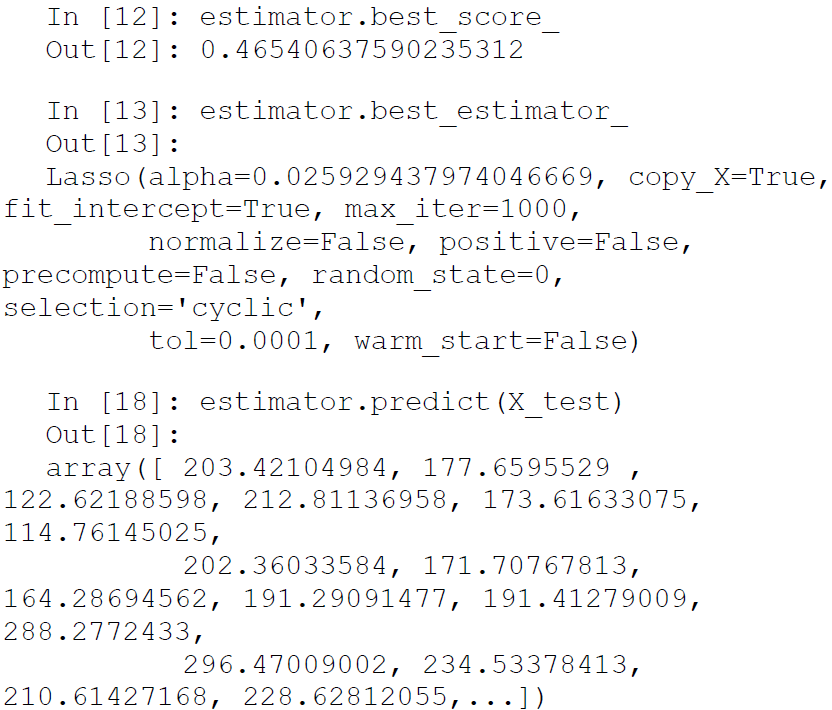
🡪Then we will define the model we want to use and the parameter space for one of the model’s hyperparameters. Here we will search the parameter alpha of the Lasso model. This parameter basically controls the strictness our regularization.



🡪Then we will initialize an estimator that will identify the model to be used. Here we notice that the process is identical for both learning a single model and a grid search of models, i.e. they both are objects of the estimator class.



🡪This will take our train set and learn a group of Lasso models by varying the value of the alpha hyperparameter. The GridSearchCV object will also score the models that we are learning and we can us the best\_estimator\_ attribute to identify the model and the optimal value of the hyperparameter that gave us the best score. Also we can directly use the same object for predicting with the best model on unknown data.



🡪The next steps involve reiterating the whole process making changes to the data transformation, Machine Learning algorithm, tuning hyperparameters of the algorithm etc., but the basic steps will remain the same. We will go into the elaborate details of these processes in future chapters of the book. Here we will conclude our introduction to the scikit-learn framework and encourage you to check out their extensive documentation at http://scikit-learn.org/stable , which points to the home page of the most current stable version of scikit-learn.

1. Neural Networks and Deep Learning]

🡪Deep learning has become one of the most well-known representations of Machine Learning in the recent years. Deep Learning applications have achieved remarkable accuracy and popularity in various fields especially in image and audio related domains. Python is the language of choice when it comes to learning deep networks and complex representations of data.

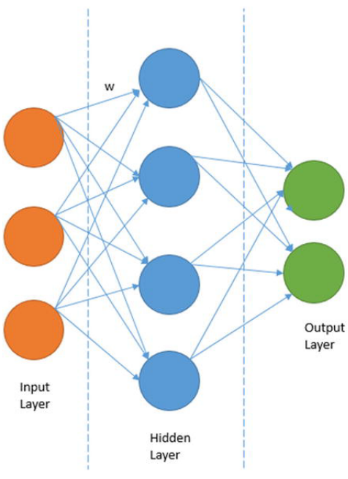
🡪In this section, we briefly discuss ANNs (Artificial Neural Networks) and Deep Learning networks. Then we will move on to the popular Deep Learning frameworks for Python. Since, the mathematics involved behind ANNs is quite advanced we will keep our introduction minimal and focused on the practical aspects of learning a neural network. We recommend you check out some standard literature on the theoretical aspects of Deep Learning and neural networks like Deep Learning by Goodfellow and Bengio, if you are more interested in its internal implementations.

🡺Artificial Neural Networks

🡪Deep learning can be considered as an extension of Artificial Neural Networks (ANNs) . Neural networks were first introduced as a method of learning by Frank Rosenblatt in 1958, although the learning model called perceptron was different from modern day neural networks, we can still regard the perceptron as the first artificial neural network.

🡪Artificial neural networks loosely work on the principle of learning a distributed distribution of data. The underlying assumption is that the generated data is a result of nonlinear combination of a set of latent factors and if we are able to learn this distributed representation then we can make accurate predictions about a new set of unknown data.

🡪The simplest neural network will have an input layer, a hidden layer (a result of applying a nonlinear transformation to the input data), and an output layer. The parameters of the ANN model are the weights of each connection that exist in the network and sometimes a bias parameter. This simple neural network is represented as shown in Figure below:



🡪This network is having an input vector of size 3, a hidden layer of size 4, and a binary output layer . The process of learning an ANN will involve the following steps.

1. Define the structure or architecture of the network we want to use. This is critical as if we choose a very extensive network containing a lot of neurons/units (each circle in Figure 2-6 can be labeled as neuron or a unit) then we can overfit our training data and our model won’t generalize well.
2. Choose the nonlinear transformation to be applied to each connection. This transformation controls the activeness of each neuron in the network.
3. Decide on a loss function we will use for the output layer. This is applicable in the case when we have a supervised learning problem, i.e. we have an output label associated with each of the input data points.
4. Learning the parameters of the neural network , i.e. determine the values of each connection weight. Each arrow in Figure above carries a connection weight. We will learn these weights by optimizing our loss function using some optimization algorithm and a method called backpropagation.

🡪We will not go into the details of backpropagation here, as it is beyond the scope of the present chapter. We will extend these topics when we actually use neural networks.

1. Deep Neural Networks

🡪Deep neural networks are an extension of normal artificial neural networks. There are two major differences that deep neural networks have, as compared to normal neural networks.

🡺**Number Of Layers** : Normal neural networks are shallow, which means that they will have at max one or two hidden layers. Whereas the major difference in deep neural networks is that they have a lot more hidden layers. And this number is usually very large. For example, the Google brain project used a neural network that had millions of neurons.

🡺**Diverse Architectures** : Based on what we discussed in Chapter 1, we have a wide variety of deep neural network architectures ranging from DNNs, CNNs, RNNs, and LSTMs. Recent research have even given us attention based networks to place special emphasis on specific parts of a deep neural network. Hence with Deep Learning, we have definitely gone past the traditional ANN architecture.

🡺**Computation Power** : The larger the network and the more layers it has, the more complex the network becomes and training it takes a lot of time and resources. Deep neural networks work best on GPU based architectures and take far less time to train than on traditional CPUs, although recent improvements have vastly decreased training times.

1. Python Libraries For Deep Learning

🡪We will learn about two packages—Theano and TensorFlow—which will allow us to build neural network based models on datasets. In addition to these we will learn to use Keras, which is a high level interface to building neural networks easily and has a concise API, capable of running on top of both TensorFlow and Theano.

🡪Besides these, there are some more excellent frameworks For Deep Learning. We also recommend you to check out PyTorch, MXNet, Caffe (recently Caffe2 was released), and Lasagne.

**🡺Theano**

🡪The first library popularly used for learning neural networks is Theano. Although by itself, Theano is not a traditional Machine Learning or a neural network learning framework, what it provides is a powerful set of constructs that can be used to train both normal Machine Learning models and neural networks. Theano allows us to symbolically define mathematical functions and automatically derive their gradient expression.

🡪This is one of the frequently used steps in learning any Machine Learning model. Using Theano, we can express our learning process with normal symbolic expressions and then Theano can generate optimized functions that carry out those steps.

🡪Training of Machine Learning models is a computationally intensive process. Especially neural networks have steep computational requirements due to both the number of learning steps involved and the non-linearity involved in them. This problem is increased manifold when we decide to learn a deep neural network.

🡪One of the important reasons of Theano being important for neural network learning is due to its capability to generate code which executes seamlessly on both CPUs and GPUs. Thus if we specify our Machine Learning models using Theano, we are also able to get the speed advantage offered by modern day GPUs.

🡪In the rest of this section, we see how we can install Theano and learn a very simple neural network using the expressions provided by Theano.

🡺**Installation**

🡪Theano can be easily installed by using the Python package manager pip or conda.



🡪Often the pip installer fails on Windows, hence we recommend using conda install theano on the Windows platform. We can verify the installation by importing our newly installed package in a Python shell.

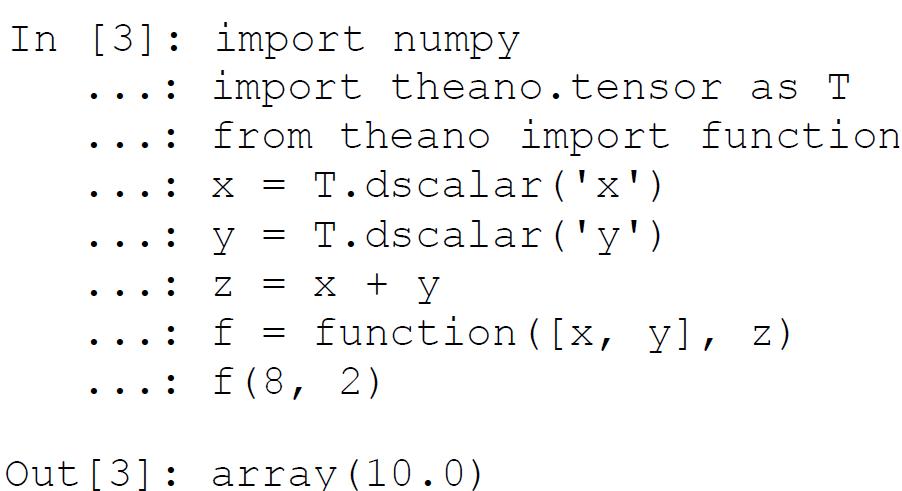


**🡺Theano Basics (Barebones Version)**

🡪In this section, we discuss some basics of the symbolic abilities offered by theano and how those can be leveraged to build some simple learning models. We will not directly use theano to build a neural network in this section, but you will know how to carry out symbolic operations in theano. Besides this, you will see in the coming section that building neural networks is much easier when we use a higher level library such as keras.

🡪**Theano expresses symbolical expressions using something called tensors**. A tensor in its simplest definition is a multi-dimensional array. So a zero-order tensor array is a scalar, a one-order tensor is a vector, and a two order tensor is a matrix.

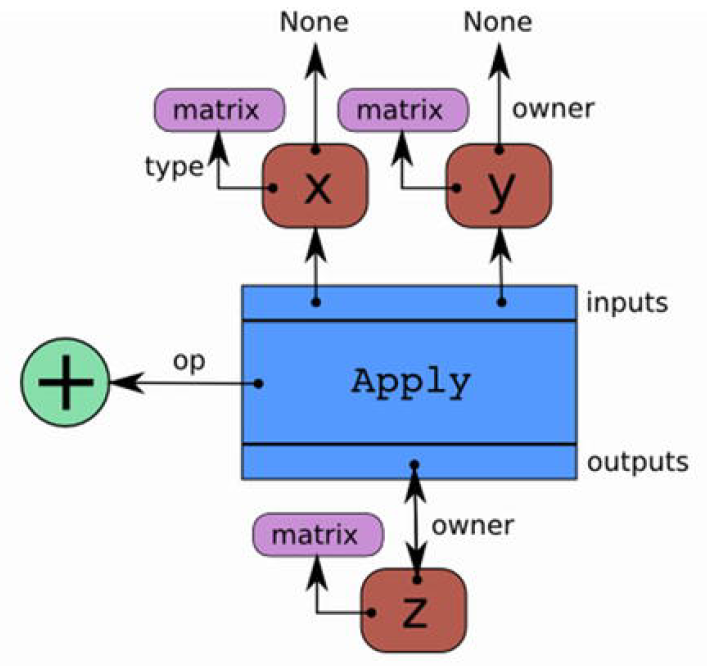
🡪Now we look at how we can work on a zero-order tensor a scalar by using constructs provided by theano.



**🡪Here, we defined a symbolical operation (denoted by the symbol z) and then bound the input and the operations in a function. This was achieved by using the function construct provided by theano.**

🡪Contrast it with the normal programming paradigm and we would need to define the whole function by ourselves. This is one of the most powerful aspects of using a symbolical mathematical package like theano. Using construct similar to these, we can define a complex set of operations.

🡺**Graph Structure** : Theano represents symbolical mathematical operations as graphs. So when we define an operation like z, as depicted in the earlier example, no calculation happens instead what we get is a graph representation of the expression. These graphs are made up of Apply, Op, and variable nodes. The Apply node represents application of some op on some set of variable nodes. So if we wanted to visualize the operation we defined in the preceding step as a graph, it would look like the depiction in Figure below



🡪**Theano has various low-level tensor APIs for building neural network architectures using Tensor arithmetic and Ops**. This is available in the theano.tensor.nnet module and you can check out relevant functions at http://deeplearning.net/software/theano/library/tensor/nnet/index.html , **which include conv for convolutional neural networks and nnet for regular neural network operations**. **This concludes our basic introduction to theano. We kept it simple because we will rarely be using theano directly and instead rely on high-level libraries like keras to build powerful deep neural networks with minimal code and focus more on solving problems efficiently and effectively**.

1. Tensorflow

🡪Tensorflow is an open source software library for Machine Learning released by Google in November 2015. Tensorflow is based on the internal system that Google uses to power its research and production systems.

🡪Tensorflow is quite similar to Theano and can be considered as Google’s attempt to provide an upgrade to Theano by providing easy-to-use interfaces into Deep Learning, neural networks, and Machine Learning with a strong focus on rapid prototyping and model deployment constructs.

🡪Li**ke Theano it also provides constructs for symbolical mathematics, which are then translated into computational graphs**. **These graphs are then compiled into lower-level code and executed efficiently**.

🡪**Like theano, tensorflow also supports CPUs and GPUs seamlessly**. In fact tensorflow works best on a **TPU**, known as the Tensor Processing Unit, which was invented by Google.

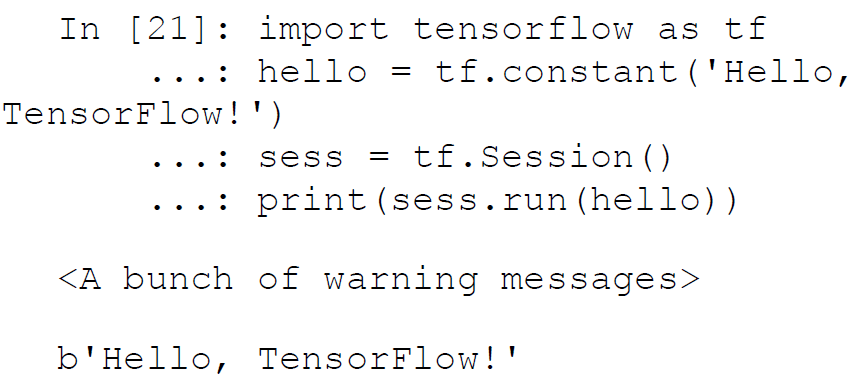
🡪In addition to having a Python API, tensorflow is also exposed by APIs to C++, Haskell, Java, and Go languages. One of the major differences tensorflow has as compared to theano is the support for higher-level operations, which ease the process of Machine Learning and its focus on model development as well as deployment to production and model serving via multiple mechanisms.

🡪Also the documentation and usage of theano is not so intuitive to use, which is another area tensorflow aims to fill, by its easy-to-understand implementations and extensive documentation.

🡺Installation

conda install tensorflow or pip install tensorflow

🡪Once we have installed the library, we can verify a successful install by verifying it in the ipython console with the following commands.



🡪The message verifies our successful install of the tensorflow library. You are also likely to see a bunch of warning messages but you can safely ignore them. The reason for those messages is the fact that the default tensorflow build is not built with support for some instruction sets, which may slow down the process of learning a bit.

1. Keras

🡪Keras is a high-level Deep Learning framework for Python, which is capable of running on top of both Theano and Tensorflow. Developed by Francois Chollet, the most important advantage of using Keras is the time saved by its easy-to-use but powerful high level APIs that enable rapid prototyping for an idea.

🡪Keras allows us to use the constructs offered by Tensorflow and Theano in a much more intuitive and easy-to-use way without writing excess boilerplate code for building neural network based models.

🡪This ease of flexibility and simplicity is the major reason for popularity of keras. In addition to providing an easy access to both of these somewhat esoteric libraries, keras ensures that we are still able to take the advantages that these libraries offer.

🡺Installation

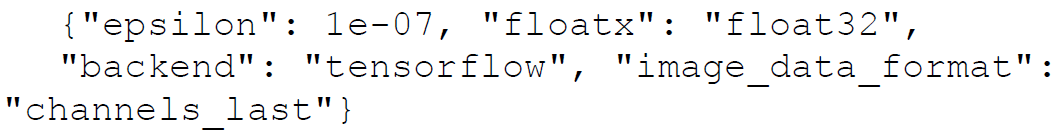
🡪Keras is easy to install using the familiar pip or conda command. We will assume that we have both tensorflow and theano installed, as they will be required to be used as backend for keras model development.

🡪And for installation of keras use:

conda install keras or pip install keras

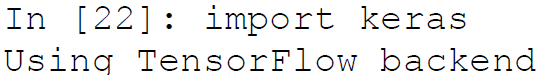
🡪We can check for the successful installation of keras in our

environment by importing it in IPython. Upon a successful import it will display the current backend, which is usually theano by default. So you need to go to the keras.json file, available under the .keras directory under your user account directory. Our config file contents are as follows.



🡪You can refer to https://keras.io/backend/ , which tells you

how easily you can switch the backend in keras from theano to tensorflow. Once the backend in specified in the config file, on importing keras, you should see the following message in your ipython shell.



🡺Keras Basics

🡪The main abstraction for a neural network is a model in keras. A model is a collection of neurons that will define the structure of a neural network. There are two different types of models:

* + **Sequential model** : Sequential models are just stacks of layers. These layers can together define a neural network.
  + **Functional API Model** : Sequential models are very useful but sometimes our requirement will exceed the constructs possible using sequential models. This is where the function model APIs will come in to the picture. This API allows us to specify complex networks i.e., networks that can have multiple outputs, networks with shared layers, etc. These kinds of models are needed when we need to use advanced neural networks like convolutional neural networks or recurrent neural networks.

🡺Model Building

🡪The model building process with keras is a three-step process. The first step is specifying the structure of the model. This is done by configuring the base model that we want to use, which is either a sequential model or a functional model. Once we have identified a base model for our problem we will further enrich that model by adding layers to the model.

🡪The model building process with keras is a three-step process. The first step is specifying the structure of the model. This is done by configuring the base model that we want to use, which is either a sequential model or a functional model. Once we have identified a base model for our problem we will further enrich that model by adding layers to the model.

🡪The next step in the model learning process is the compilation of the model architecture that we defined in the first step. Based on what we learned in the preceding sections on Theano and Tensorflow, **most of the model building steps are symbolic and the actual learning is deferred until later**. In the compilation step, we configure the learning process. **The learning process, in addition to the structure of the model, needs to specify the following additional three important parameters**:

* + **Optimizer** : We learned in the first chapter that the simplest explanation of a learning process is the optimization of a loss function. Once we have the model and the loss function, we can specify the optimizer that will identify the actual optimization algorithm or program we will use, to train the model and minimize the loss or error. This could be a string identifier to the already implemented optimizers, a function, or an object to the Optimizer class that we can implement.
  + **Loss Function** : A loss function, also known as an objective function, will specify the objective of minimizing loss/error, which our model will leverage to get the best performance over multiple epochs/iteration. It again can be a string identifier to some preimplemented loss functions like cross-entropy loss (classification) or mean squared error (regression) or it can be a custom loss function that we can develop.
  + **Performance Metrices** : A metric is a quantifiable measure of the learning process. While compiling a model, we can specify a performance metric we want to track (for example, accuracy for a classification model), which will educate us about the effectiveness of the learning process. This helps in evaluating model performance.

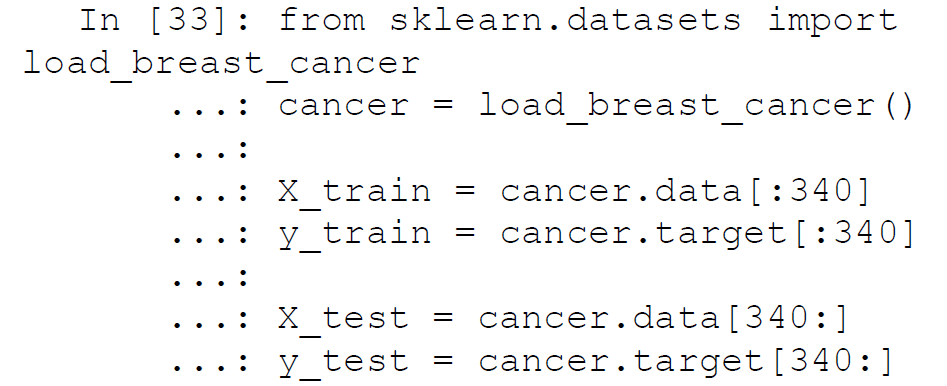
🡪 The last step in the model building process is executing the compiled method to start the training process. This will execute the lower level compiled code to find out the necessary parameters and weights of our model during the training process.

🡪In keras, like scikit-learn, it is achieved by calling the fit function on our model. We can control the behavior of the function by supplying appropriate arguments. You can learn about these arguments at https://keras.io/models/sequential/ .

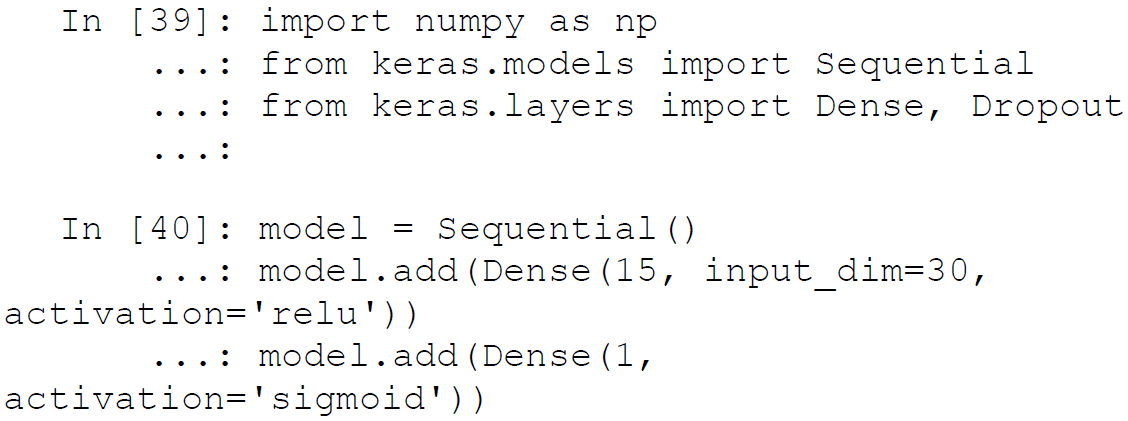
1. Learning an Example Neural Network

🡪 We will conclude this section by building a simple working neural network model on one of the datasets that comes bundled with the scikit-learn package. We will use the tensorflow backend in our example, but you can try to use a theano backend and verify the execution of model on both the backends.

🡪 For our example, we will use the Wisconsin Breast Cancer dataset, which is bundled with the scikit-learn library. The dataset contains attribute drawn from a digitized image of fine needle aspirate of a breast mass. They describe characteristics of the cell nuclei present in the image. On the basis of those attributes, the mass can be marked as malignant or benign. The goal of our classification system is to predict that level. So let’s get started by loading the dataset.



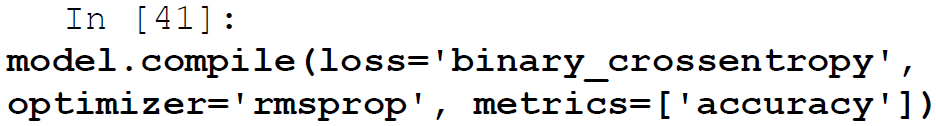
🡪 The next step of the process is to define the model architecture using the keras model class. We see that our input vector is having 30 attributes so we will have a shallow network having one hidden layer of half the units (neurons), i.e., we will have 15 units in the hidden layer. We add a one unit output layer to predict either 1 or 0 based on whether the input data point is benign or malignant. This is a simple neural network and doesn’t involve Deep Learning.



🡪 Here we have defined a sequential keras model , which is having a

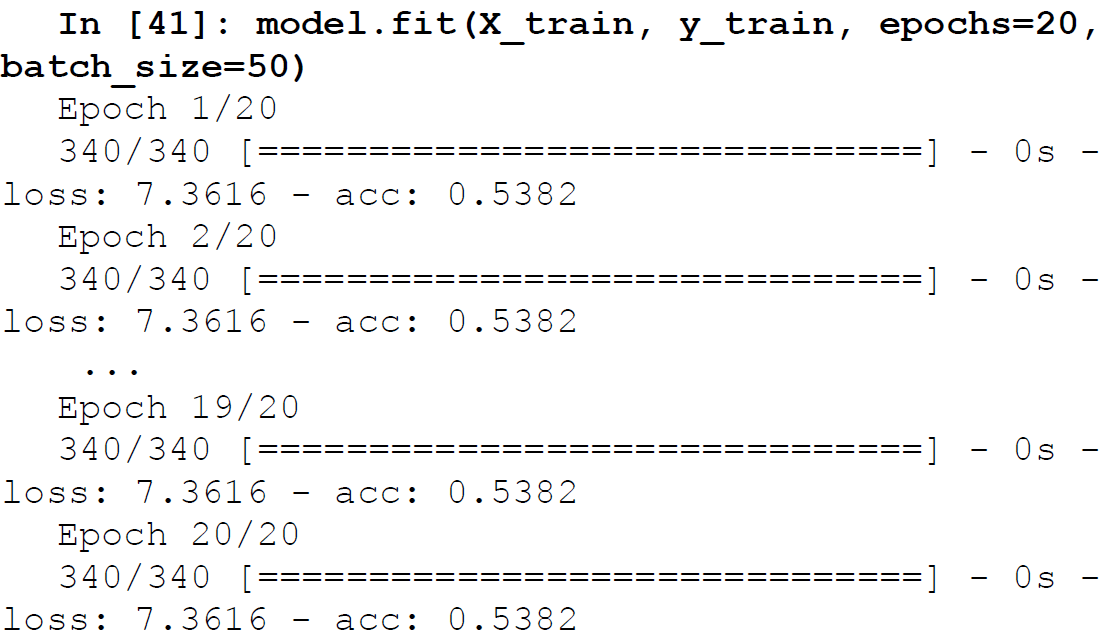
dense hidden layer of 15 units. The dense layer means a fully connected

layer so it means that each of those 15 units (neurons) is fully connected to the 30 input features. The output layer for our example is a dense layer with the sigmoid activation. The sigmoid activation is used to convert a real valued input into a binary output (1 or 0). Once we have defined the model we will then compile the model by supplying the necessary optimizer, loss function, and the metric on which we want to evaluate the model performance.



🡪 Here we used a loss function of binary\_crossentropy, which is a

standard loss function for binary classification problems. For the optimizer, we used rmsprop, which is an upgrade from the normal gradient descent algorithm. The next step is to fit the model using the fit function.



🡪 Here, the epochs parameter indicates one complete forward and backward pass of all the training examples.

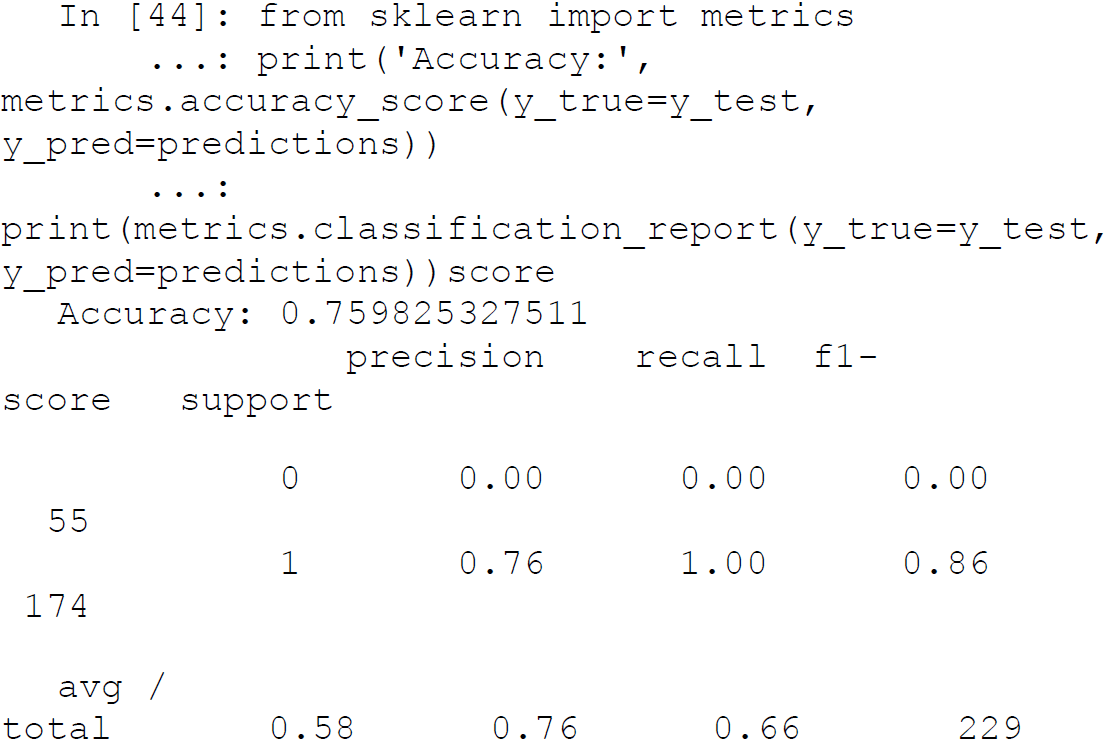
🡪 The batch\_size parameter indicates the total number of samples which are propagated through the NN model at a time for one backward and forward pass for training the model and updating the gradient.

🡪 Thus if you have 100 observations and your batch size is 10, each epoch will consist of 10 iterations where 10 observations (data points) will be passed through the network at a time and the weights on the hidden layer units will be updated. However we can see that the overall loss and training accuracy remains the same. Which means the model isn’t really learning anything from the looks of it!

🡪 The API for keras again follows the convention for scikit-learn models, hence we can use the predict function to predict for the data points in the test set. In fact we use predict\_classes to get the actual class label predicted for each test data instance.



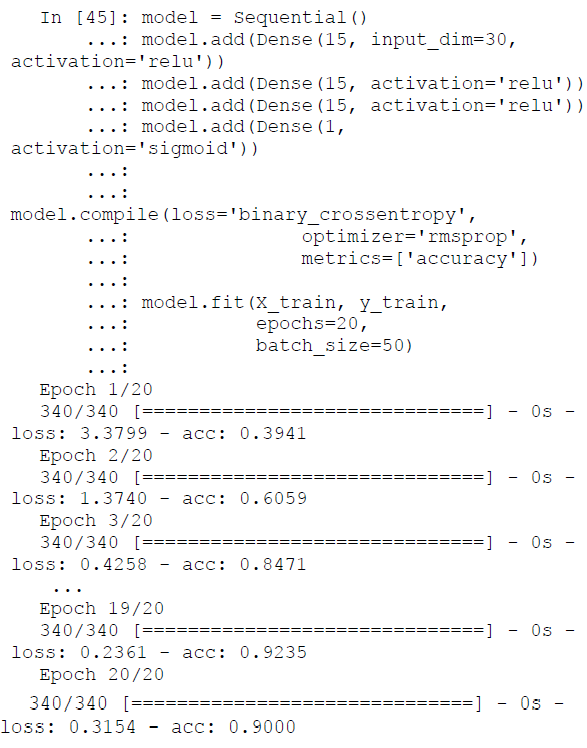
🡪 Let’s evaluate the model performance by looking at the test data accuracy and other performance metrics like precision, recall, and F1 score. Do not despair if you do not understand some of these terms, as we will be covering them in detail in Chapter 5. For now, you should know that scores closer to 1 indicate better results i.e., an accuracy of 1 would indicate 100% model accuracy, which is perfection. Luckily, scikit-learn provides us with necessary performance metric measuring APIs.



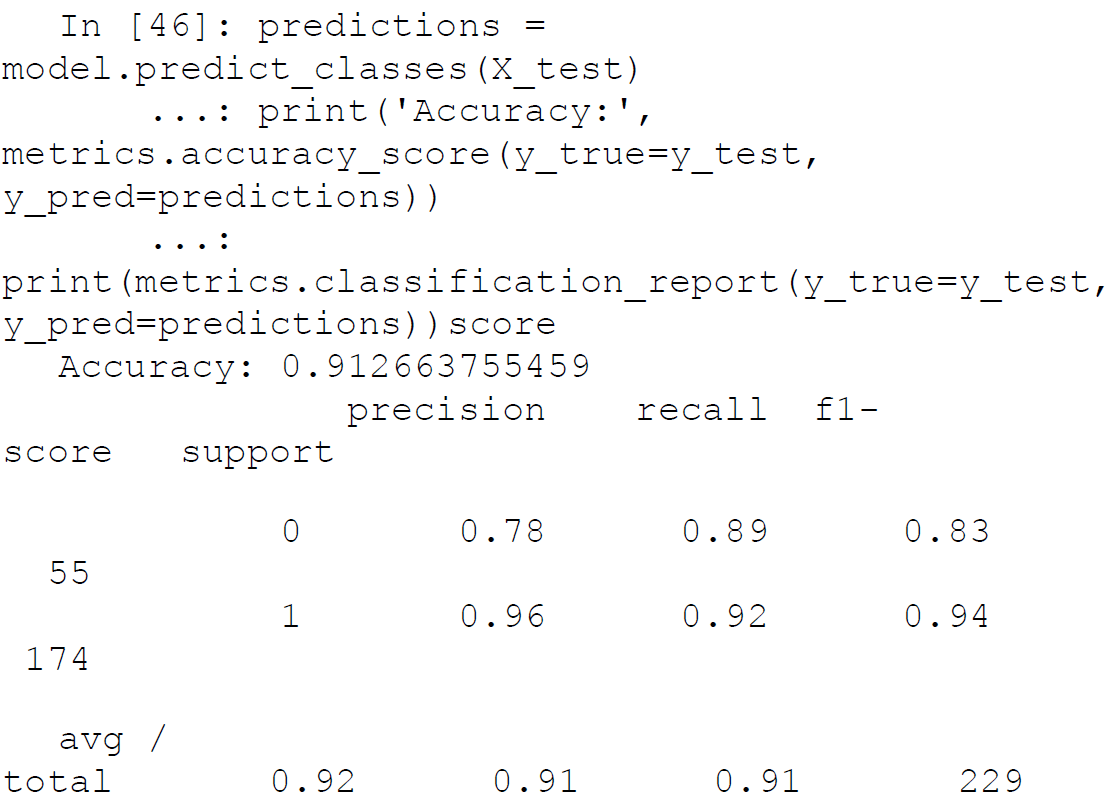
🡪 From the previous performance metrics , we can see that even though model accuracy is 76%, for data points having cancer (malignant) i.e., label 0, it misclassifies them as 1 (55 instances) and remaining 174 instances where class label is 1 (benign), it classifies them perfectly. Thus this model hasn’t learned much and predicts every response as benign (label 1). Can we do better than this?

1. The Power Of Deep Learning

🡪 The idea of Deep Learning is to use multiple hidden layers to learn latent and complex data patterns, relationships, and representations to build a model that learns and generalizes well on the underlying data. Let’s take the previous example and convert it to a fully connected deep neural network (DNN) by introducing two more hidden layers. The following snippet builds and trains a DNN with the same configuration as our previous experiment only with the addition of two new hidden layers.



🡪 We see a remarkable jump in the training accuracy and a drop in the loss based on the preceding training output. This is indeed excellent and seems promising! Let’s check out our model performance on the test data now.



🡪 We achieve an overall accuracy and F1 score of 91% and we can see that we also have an F1 score of 83% as compared to 0% from the previous model, for class label 0 (malignant). Thus you can clearly get a feel of the power of Deep Learning, which is evident by just introducing more hidden layers in our network, which enabled our model to learn better representations of our data. Try experimenting with other architectures or even introducing regularization aspects like dropout.

🡪 Thus, in this section, you learned about some of the important

frameworks relevant to neural networks and Deep Learning. We will revisit the more advanced aspects of these frameworks in subsequent chapters when we work on real-world case studies.

1. Text Analytics and Natural Language Processing

🡪 In the sections till now we have mostly dealt with structured data formats and datasets i.e., data in which we have the observations occurring as rows and the features or attributes for each of those observations occurring as columns. This format is most convenient for Machine Learning algorithms but the problem is that raw data is not always available in this easy-to interpret format. This is the case with unstructured data formats like audio, video, textual datasets. In this section, we try to get a brief overview of the frameworks we can use to solve this problem if the data that we are working with is unstructured text data. We will not go into detailed examples of using these frameworks and if you are interested, we recommend checking out Chapter 7 of this book, which deals with a realworld case study on analyzing text data.

1. The Natural Language Tool Kit

🡪 Perhaps the most important library of Python to work with text data is NLTK or the Natural Language Tool Kit. This section introduce NLTK and its important modules. We go over the installation procedure of the library and a brief description of its important modules.

🡺Installation and Introduction

🡪 The nltk package can be installed in the same way as most of the other packages used in this book, which is by using the pip or conda command.

**conda install nltk**

🡪 We can verify the installation by importing the package in an

IPython/Python shell.



🡪 There’s an important difference for the nltk library as compared to other standard libraries. In case of other libraries, in general, we don’t need to download any auxiliary data. But for the nltk library to work to its full potential, we would require some auxiliary data, which are mostly various corpora. This data is leveraged by multiple functions and modules in the library. We can download this data by executing the following command in

the Python shell.



🡪 This command will give us the screen shown in Figure below, where we can select the additional data we want to install and select the installation location. We will select to install all the additional data and packages available.



🡪 You can also choose to download all necessary datasets without the GUI by using the following command from the ipython or Python shell.



🡪 Once the download is finished we will be able to use all the necessary functionalities and the bundled data of the nltk package. We will now take a look at the major modules of nltk library and introduce the functionality that each of them provides.

**🡺Copra**

🡪 The starting point of any text analytics process is the process of collecting the documents of interest in a single dataset. This dataset is central to the next steps of processing and analysis. **This collection of documents is**

**generally called a corpus**. Multiple corpus datasets are called corpora. The nltk module nltk.corpus provides necessary functions that can be used to read corpus files in a variety of formats. It supports the reading of corpora from the datasets bundled in nltk package as well as external corpora.

🡺Tokenization

🡪 Tokenization is one of the core steps in text pre-processing and normalization. Each text document has several components like paragraphs, sentences, and words that together make up the document. The process of tokenization is used to break down the document into these smaller components.

🡪 This tokenization can be into sentences, words, clauses, and so on. The most popular way to tokenize any document is by using sentence tokenization and\or word tokenization. The nltk.tokenize module of the nltk library provides functionality that enables efficient tokenization of any textual data.

🡺Tagging

🡪 A text document is constructed based on various grammatical rules and constructs. The grammar depends on the language of the text document. Each language’s grammar will contain different entities and parts of speech like nouns, pronouns, adjectives, adverbs, and so on.

🡪 The process of tagging will involve getting a text corpus, tokenizing the text and assigning metadata information like tags to each word in the corpora. The nltk.tag module contains implementation of different algorithms that can be used for such tagging and other related activities.

🡺Stemming and Lemmatization

🡪 A word can have several different forms based on what part of speech it is representing. Consider the word fly; it can be present in various forms in the same text, like flying, flies, flyer, and so on. The process of stemming is used to convert all the different forms of a word in to the base form, which is known as the root step. Lemmatization is similar to stemming but the base form is known as the root word and it’s always a semantically and lexicographically correct word.

🡪 This conversion is crucial, as a lot of times the core word contains more information about the document, which can be diluted by these different forms. The nltk module nltk.stem contains different techniques that can be used for stemming and lemmatizing a corpus.

🡺Chunking

🡪Chunking is a process which is similar to parsing or tokenization but the major difference is that instead of trying to parse each word, we will target phrases present in the document. Consider the sentence “The brown fox saw the yellow dog”. In this sentence, we have two phrases which are of interest. The first is the phrase “the brown fox,” which is a noun phrase and the second one is the phrase “the yellow dog,” which again is a noun phrase. By using the process of chunking, we are able to tag phrases with additional parts of speech information, which is important for understanding the structure of the document. The nltk module nltk.chunk consists of necessary techniques that can be used for applying the chunking process to our corpora.

🡺Sentiment

🡪Sentiment or emotion analysis is one of the most recognizable applications on text data. Sentiment analysis is the process of taking a text document and trying to determine the opinion and polarity being represented by that document. Polarity in the reference of a text document can mean the emotion, e.g., positive, negative, or neutral being represented by the data. The sentiment analysis on textual data can be done using different algorithms and at different levels of text segmentation. The nltk.sentiment package is the module that can be used to perform different sentiment analyses on text documents. Check out Chapter 7 for a real-world case study on sentiment analysis!\

🡺Classification/Clustering

🡪Classification of text documents is a supervised learning problem, as we explained in the first chapter. Classification of text documents may involve learning the sentiment, topic, theme, category, and so on of several text documents (corpus) and then using the trained model to label unknown documents in the future. The major difference from normal structured data comes in the form of feature represent-tations of unstructured text we will be using.

🡪Clustering involves grouping together similar documents based on some similarity measure, like cosine similarity, bm25 distance, or even semantic similarity. The nltk.classify and nltk.cluster modules are typically used to perform these operations once we do the necessary feature engineering and extraction.

1. Other Text Analysis Frameworks

🡪Typically, nltk is our go-to library for dealing with text data, but the Python ecosystem also contains other libraries that can be useful in dealing with textual data. We will briefly mention some of these libraries so that you get a good grasp of the toolkit that you can arm yourself with when dealing with unstructured textual data.

* **patten** : The pattern framework is a web mining module for the Python programming language. It has tools for web mining (extracting data from Google, Twitter, a web crawler, or an HTML DOM parser), information retrieval, NLP, Machine Learning, sentiment analysis and network analysis, and visualization. Unfortunately, pattern currently works best on Python 2.7 and there is no official port for Python 3.x.
* **genism** : The gensim framework, which stands for generate similar, is a Python library that has a core purpose of topic modeling at scale! This can be used to extract semantic topics from documents. The focus of gensim is on providing efficient topic modeling and similarity analysis. It also contains a Python implementation of Google’s popular word2vec model.
* **textblob** : This is another Python library that promises simplified text processing. It provides a simple API for doing common text processing tasks including parts of speech tagging, tokenization, phrase extraction, sentiment analysis, classification, translation, and much more!
* **spacy** : This is a recent addition to the Python text processing landscape but an excellent and robust framework nonetheless. The focus of spacy is industrial strength natural language processing, so it targets efficient text analytics for large-scale corpora. It achieves this efficiency by leveraging carefully memory-managed operations in Cython. We recommend using spacy for natural language processing and you will also see it being used extensively for our text normalization process in Chapter 7.

1. Statsmodels

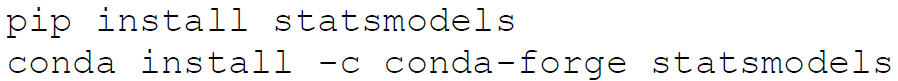
🡪Statsmodels is a library for statistical and econometric analysis in Python. The advantage of languages like R is that it’s a statistically focused language with lot of capabilities. It consists of easy-to-use yet powerful models that can be used for statistical analysis and modeling. However from deployment, integration, and performance aspects, data scientists and engineers often prefer Python but it doesn’t have the power of easy-to-use statistical functions and libraries like R.

🡪The statsmodels library aims to bridge this gap for Python users. It provides the capabilities for statistical, financial and econometric operations with the aim of combining the advantages of Python with the statistical powers of languages like R. Hence users familiar with R, SAS, Stata, SPSS, and so on who might want similar functionality in Python can use statsmodels.

🡺Installation

🡪The package can be installed using pip or conda install and the

following commands.



🡺Modules

🡪In this section, we briefly cover the important modules that comprise the statsmodel package and the capability those models provides. This should give you enough idea of what to leverage to build statistical models and perform statistical analysis and inference.

🡺**Distributions** : One of the central ideas in statistics is the distributions of statistical datasets. Distributions are a listing or function that assigns a probability value to all the possible values of the data. The distributions module of the statsmodels package implements some important functions related to statistical distribution including sampling from the distribution, transformations of distributions, generating cumulative distribution functions of important distributions, and so on.

🡺**Linear Regression** : Linear regression is the simplest form of statistical modeling for modeling the relationship between a response dependent variable and one or more independent variables such that the response variable typically follows a normal distribution. The statsmodels.regression module allows us to learn linear models on data with IID i.e., independently and identically distributed errors. This module allows us to use different methods like ordinary least squares (OLS), weighted least squares (WLS), generalized least squares (GLS), and so on, for the estimation of the linear model parameters.

🡺**Generalized Linear Model** : Normal linear regression can be generalized if the dependent variable follows a different distribution than the normal distribution. The statsmodels.genmod module allows us to extend the normal linear models to different response variables. This allows us to predict the linear relationship between the independent and dependent variable when the dependent variable follows distributions other than normal distributions.

🡺**ANOVA** : Analysis of variance is a process of statistical processes used to analyze the difference between group means and associated procedures. ANOVA analysis is an important way to test whether the means of several groups are equal or unequal. This is an extremely powerful tool in hypothesis testing and statistical inference and is implemented in the anova\_lm module of the statsmodel package.

🡺**Time Series Analysis** : Time series analysis is an important part of data analytics. A lot of data sources like stock prices, rainfall, population statistics, etc. are periodic in nature. Time series analysis is used find structures, trends, and patterns in these streams of data. These trends can be used to understand the underlying phenomena using a mathematical model and even make predictions and forecasts about future events

🡪Basic time series models include univariate autoregressive models (AR), vector autoregressive models (VAR), univariate autoregressive moving average models (ARMA), as well as the very popular autoregressive integrated moving average (ARIMA) model. The tsa module of the statsmodels package provides implementation of time series models and also provides tools for time series data manipulation.

🡺**Statistical Inference** : An important part of traditional statistical inference is the process of hypothesis testing. A statistical hypothesis is an assumption about a population parameter. Hypothesis testing is the formal process of accepting or rejecting the assumption made about the data on the basis of observational data collected from samples taken from the population.

🡪The stats.stattools module of statsmodels package implements the most important of the hypothesis tests. Some of these tests are independent of any model, while some are tied to a particular model only.

🡺**Nonparametric Methods** : Nonparametric statistics refers to statistics that is not based on any parameterized family of probability distributions. When we make an assumption about the distribution of a random variable we assign the number of parameters required to ascertain its behavior. For example, if we say that some metric of interest follows a normal distribution it means that we can understand its behavior if we are able to determine the mean and variance of that metric. This is the key difference in non-parametric methods, i.e., we don’t have a fixed number of parameters that are required to describe an unknown random variable. Instead the number of parameters are dependent on the amount of training data. The module nonparametric in the statsmodels library will help us perform non-parametric analysis on our data.