Hyperiondev

TASK

# Python Packages for Data Science

Visit our website

# Introduction

## WELCOME TO THE PYTHON PACKAGES FOR DATA SCIENCE TASK!

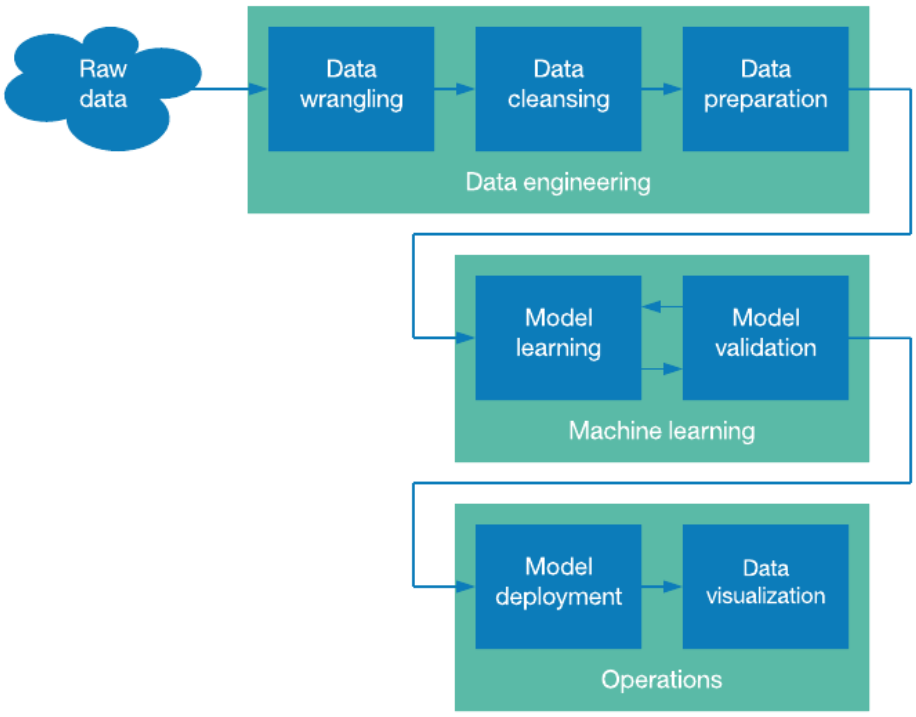
Python, along with R, is one of the most commonly used data science tools. Python offers a range of active data science libraries and a vibrant user community, which makes it a valuable asset for data scientists. In this task, you will learn about some of the most popular Python libraries and packages for Data Science.

**Take note:**

A **package** is a collection of related modules (Python files) that enable some particular functionality. A **library** is a more broad term meaning a bundle of code. These terms are often used interchangeably (**Source**).

## THE DATA SCIENCE PIPELINE

There are various tasks that data scientists perform to analyse data in a meaningful way. Different people describe these steps differently. The steps don't have to be carried out in a strict order. However, the diagram below broadly shows the steps that are usually involved in the data science pipeline:




(Source: Jones, 2018)

Once you have obtained your raw data the next step is: data engineering. There are three main tasks in data engineering:

- Data wrangling
- Data cleaning
- Data preparation

**Data wrangling** is defined as "the process by which you identify, collect, merge, and preprocess one or more data sets in preparation for data cleansing" (Jones, 2018). Once you've wrangled the data into a format with which you can work, you **clean** the data by fixing any errors in the dataset. This could involve dealing with missing data, correcting formatting issues and fixing, or removing, incorrect data. You can then prepare the data for machine learning. Various visualisations (such as graphs, tables and charts) are also used to gain this insight into the data during this phase.

During the machine learning step, you will build and validate models to gain insight into the data.

Finally, the operations step involves deploying machine learning models and producing visualisations to present the findings of the models. Data visualisation is important at both ends of the data science pipeline to present the findings of a study and to help gain an understanding of data at the start. Deploying models is outside the scope of this bootcamp, but the visualisation techniques you will learn about will be relevant to this phase of the data science pipeline.

For each of the steps in the data science pipeline, there are third-party Python packages and libraries that you can leverage to perform the tasks in each step. Packages contain a collection of code files that have already been written so you don't need to reinvent the wheel to achieve certain functionality in your program.

All the third-party packages you need for this bootcamp were installed at the beginning of the bootcamp. However, if you would like to install additional packages you are welcome to do so.

**Extra resource**

Explore the **additional reading** in this task folder for best practices when installing packages. You already have the relevant package managers installed from the environment setup at the beginning of the bootcamp.

A note from our coding mentor

**Melanie**

If you have pip installed, you can install many different libraries available in Python! There are libraries such as **Pillow**, which helps you with image manipulation, and **Arcade** to create 2D games. My personal favourite Python library is **Scrapy**; it can extract and collect data from web pages and web APIs!

## IMPORTING PACKAGES

To use packages or libraries in Python, you will need to import them at the top of your Python file. There are several ways to achieve this. We will use the NumPy package to illustrate three methods.

1. The standard approach uses a simple `import` statement.

```
import numpy
```

   This will create a reference to the NumPy module in the current namespace. To access NumPy functions use `numpy.X`, where X is a NumPy function. For example, `numpy.array()`:

```
# Converts a list of float values into an array
numpy_values = numpy.array([22.5, 24.9, 27.8, 27.1] )
```

2. When several NumPy functions are to be used, we use a placeholder or nickname instead of NumPy by adding `as "nickname"` to the end of the import statement. This is the most commonly used option.

```
import numpy as np
```

When you need to invoke something from NumPy, use `nickname.X` instead of `numpy.X`. For example, `np.array()`:

```
# Converts a list of float values into an array
numpy_values = np.array([22.5, 24.9, 27.8, 27.1])
```

3. To avoid the dot notation – i.e. `np.X`, or any reference to NumPy itself – in your script, you can simply import all the methods (functions) from NumPy. The asterisk (*) denotes "everything", so this will import 'everything' from NumPy. Most of the time it is better to use one of the previous two options to keep a connection between the module and its functions as other modules may have functions with the same name.

```
from numpy import *
```

For example, `array()`:

```
# Converts a list of float values into an array
numpy_values = array([22.5, 24.9, 27.8, 27.1])
```

**Extra resource**

You can also import single functions or submodules. Explore **additional methods for importing third-party** code.

## PYTHON PACKAGES FOR DATA SCIENCE

During the remainder of the bootcamp, you will learn many techniques needed for data engineering, data visualisation, and machine learning. It is essential to understand and be able to use Python packages to achieve these tasks. Here you will be briefly introduced to some key data science packages that you will leverage in upcoming tasks.

## NUMPY

NumPy stands for Numeric Python. It is an open-source extension module for Python and hence uses Python syntax. NumPy is a fundamental package for scientific computing with Python – it provides common mathematical and numerical routines in the form of pre-compiled functions.

NumPy brings the power of data structures to Python. Due to its array-oriented programming, it is easy to use NumPy to work with large multidimensional arrays and matrices and to carry out scientific computations.

Here's a simple example comparing NumPy and core Python:

```
import numpy as np
# Convert a list of values in degrees Celsius to Fahrenheit
c_values = [22.5, 24.9, 27.8, 27.1]

# Create a numpy array and use numpy scalar multiplication
f_values_numpy = np.array(c_values) * 9 / 5 + 32
print(f_values_numpy) # Outputs array: [ 72.5 76.82 82.04 80.78]


# Using core Python iterate through all items in c_values
# and multiply each value by 9, divide by 5, and add 32
f_values_core = [x * 9 / 5 + 32 for x in c_values]
print(f_values_core) # Outputs list: [ 72.5, 76.82, 82.04, 80.78]
```

You might wonder what the difference is. The main benefits of using NumPy arrays are smaller memory consumption, better runtime, and greater ease of manipulation of data compared to Python. As you can see in the code above, you don't have to loop through a NumPy array!

**Extra resource**

**Learn more about Numpy** and delve into additional examples in Chapter 4 of *Python for Data Analysis*, 2nd Edition by Wes McKinney. You may also consult the **NumPy documentation** to explore its functionality.

## SCIPY

**SciPy** is a Python library that "provides many user-friendly and efficient numerical routines, such as routines for numerical integration, interpolation, optimization, linear algebra, and statistics" (SciPy.org, 2020). Learn more about the SciPy package through its **documentation**.

## PANDAS

**pandas** (yes, with a lowercase 'p') is a Python module that contains high-level data structures and tools designed for fast and easy data analysis. Pandas also provides for explicit data alignment, which prevents common errors that result from misaligned data coming in from different sources. All this makes pandas arguably the best tool for data wrangling.

The common convention for importing pandas is as follows:

```
import pandas as pd
```

Before we go further, it is important to understand the three fundamental pandas data structures: the Series (one-dimensional), DataFrame (two-dimensional), and Index.

### Series

A `Series` is a one-dimensional labelled array that can hold any data type (integers, strings, floating-point numbers, Python objects, etc.).

On creating a `Series` by passing a list of values, pandas creates a default integer index (**pandas.Index**). We can create a `Series` in pandas as shown below.

```
import pandas as pd
import numpy as np

s = pd.Series([1, 3, 5, np.nan, 6, 8])
print(s)
```

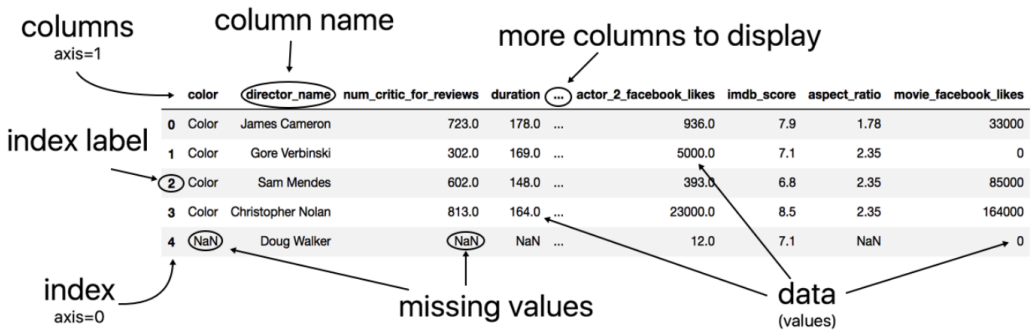
Note the use of `np.nan` to create a NaN value which is used to represent missing values or other undefined entries.

```
Output:
0    1.0
1    3.0
2    5.0
3    NaN
4    6.0
5    8.0
dtype: float64
```

**DataFrame**

You can think of a DataFrame as a table of data with the following characteristics (Lynn, 2018):

- "There can be multiple rows and columns in the data.
- Each row represents a sample of data.
- Each column contains a different variable that describes the samples (rows).
- The data in every column is usually the same type of data – e.g. numbers, strings, dates.
- Usually, unlike an Excel dataset, DataFrames avoid having missing values, and there are no gaps and empty values between rows or columns."




Anatomy of a DataFrame

*Image source: (Petrou, 2017)*

You can create a Dataframe by passing a NumPy array. In the following example, the datetime data type is specified as the index and the column labels are A, B, C, and D:

```
import pandas as pd
import numpy as np

# Generate dates for 6 different days
dates = pd.date_range('20130101', periods=6)

# Create DataFrame with dates at the index column
df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))
print(df)
```

```
Output:
                   A         B         C         D
2013-01-01  0.031025 -0.285793 -0.024709  0.400670
2013-01-02  0.713959 -1.356312 -0.581539 -0.591893
2013-01-03 -2.009914 -2.153678  0.092108  2.419320
2013-01-04 -1.521448 -1.475796 -0.695689 -0.282321
2013-01-05 -0.975965 -0.853425 -0.451579 -1.724022
2013-01-06 -0.241997 -0.433157  1.082101 -0.986914
```

Or, create a DataFrame by passing a dictionary of objects:

```
# Create DataFrame with a dictionary of objects
df2 = pd.DataFrame({ 'A' : 1.,
                     'B' : pd.Timestamp('20130102'),
                     'C' : pd.Series(1,index=list(range(4)), dtype='float32'),
                     'D' : np.array([3] * 4,dtype='int32'),
                     'E' : pd.Categorical(["test","train","test","train"]),
                     'F' : 'foo' })

# Get the data types for each column
print(df2.dtypes)
```

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

Or you could load data from a csv file into a DataFrame using the `read_csv()` function as shown below:

```
pd.read_csv('credit.csv', delimiter = ',')
```

See the example files that accompany this task to see how the statement above is used to read data from a file called **credit.csv**.

Common pandas methods and properties to explore data include:

- `head()`: returns the first 5 rows e.g. `df.head()`
- `tail()`: returns the last 5 rows e.g. `df.tail()`
- `sample()`: returns a random sample of rows, the default is 1 e.g. `df.sample(3)`
- `describe()`: returns a statistic summary for your data e.g. `df.describe()`
- `index`: stores the row labels (index) of the DataFrame e.g. `df.index`
- `columns`: stores the column of the DataFrame e.g. `df.columns`
- `values`: returns a NumPy array of the DataFrame e.g. `df.values`

Explore the **pandas documentation** to leverage all the methods and properties it provides. Part of data exploration is visualising the data, we'll get into libraries to achieve this next.

## MATPLOTLIB

There are many packages in Python that allow one to perform powerful data analysis. **Matplotlib** is "... a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, Jupyter notebook, web application servers, and four graphical user interface toolkits" (Hunter, 2007). With matplotlib, we are able to draw many different graphs in Python. Keep the **matplotlib documentation** in mind when you get stuck or want to try a new type of visualisation.

### Plotting with pandas

Although pandas is not a data visualisation package it can be used to create basic plots to simplify the pipeline during data exploration. The **pandas .plot() method** uses matplotlib as the default backend package and can be used in conjunction with matplotlib.

## SEABORN

Seaborn is a data visualisation library that has been built on top of matplotlib. While matplotlib provides basic graphs, such as line and bar charts, Seaborn can provide a bit more in terms of graphing. In addition, it integrates quite well with pandas. You may explore the **seaborn documentation** now or later when we introduce seaborn in more detail.

## SKLEARN

Scikit-learn (or sklearn) is a nice and simple starting point for using machine learning. It comes bundled with many techniques, such as preprocessing, which is used during data preparation. However, it also comes bundled with many of the basic machine learning algorithms. It also has a simple and intuitive interface: simply **instantiate** your model, **fit** it to the data, and **make a prediction**. It also includes methods for calculating accuracy, loss, etc.!

## SPACY

**SpaCy** is a Python natural language processing library (NLP) that helps you to handle large volumes of text to create NLP applications. It provides functionality to extract insights such as customer sentiment about a company or identify similar search terms to recommend similar products.

There is a lot to learn about Data Science libraries! This task has only scratched the surface.

**Extra resource**

If you would like more information about working with some of these libraries, consult the book, "**Python Data Science Handbook**" by Jake VanderPlas (Chapters 2 and 3) for more information.

# Instructions

First, read and run the **example files** using VS Code. Feel free to write and run your own example code before doing the compulsory task to become more comfortable with the concepts covered in this task.

# Compulsory Task 1

- Create a new program named **numpy_task.py**. Add comments and code to answer the following questions:
- Why doesn't `np.array((1, 0, 0), (0, 1, 0), (0, 0, 1, dtype=float)` create a two-dimensional array? Write it the correct way (Hint: Get familiar with **NumPy arrays**).
- What is the difference between `a = np.array([0, 0, 0])` and `a = np.array([[0, 0, 0]])`?
- A 3 by 4 by 4 array is created with `arr = np.linspace(1, 48, 48).reshape(3, 4, 4)`. Index or slice this array to obtain the following:
    - 20.0
    - [ 9. 10. 11. 12.]
    - [[33. 34. 35. 36.] [37. 38. 39. 40.] [41. 42. 43. 44.] [45. 46. 47. 48.]]
    - [[5. 6.], [21. 22.] [37. 38.]]
    - [[36. 35.] [40. 39.] [44. 43.] [48. 47.]]
    - [[13. 9. 5. 1.] [29. 25. 21. 17.] [45. 41. 37. 33.]]
    - [[1. 4.] [45. 48.]]
    - [[25. 26. 27. 28.], [29. 30. 31. 32.], [33. 34. 35. 36.], [37. 38. 39. 40.]]
    - Hint: Look into these NumPy tools
        - **Array manipulation routines**
        - **numpy.linspace**
        - **numpy.reshape**
        - **numpy.ndarray.flatten**

Rate us

## Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved? Do you think we've done a good job?

**Click here** to share your thoughts anonymously.

## REFERENCES


Jones, M. T. (2018, February 1). Data, structure, and the data science pipeline. Retrieved May 20, 2019, from IBM developer: **https://developer.ibm.com/articles/ba-intro-data-science-1/**

Lynn, S. (2018). The Pandas DataFrame – loading, editing, and viewing data in Python. Retrieved from Shane Lynn: Pandas Tutorials: **https://www.shanelynn.ie/using-pandas-dataframe-creating-editing-viewing-data-in-python/**

MeteoInfo. (2018, August 30). Series and DataFrame. Retrieved April 16, 2019, from meteothink.org: **http://meteothink.org/docs/meteoinfolab/userguide/dataframe.html**

Package management basics—Learn web development | MDN. (2023, May 8). **https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Understanding_client-side_tools/Package_management**

pandas.pydata.org. (n.d.). pandas.DataFrame. Retrieved April 16, 2019, from pandas 0.24.2 documentation: **https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html**

Petrou, T. (2017, October 27). Dissecting the anatomy of a DataFrame. (Packt>) Retrieved from Pandas Cookbook: Recipes for Scientific Computing, Time Series Analysis and Data Visualization using Python: **https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781784393878/1/ch01lvl1sec12/dissecting-the-anatomy-of-a-dataframe**

SciPy.org. (2020). SciPy library — SciPy.org. Retrieved 18 August 2020, from **https://svn.scipy.org/scipylib/index.html**