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, we are introduced to three of the most popular Python packages for Data Science, namely: NumPy, SciPy and Pandas.

Get in touch

Connect for support

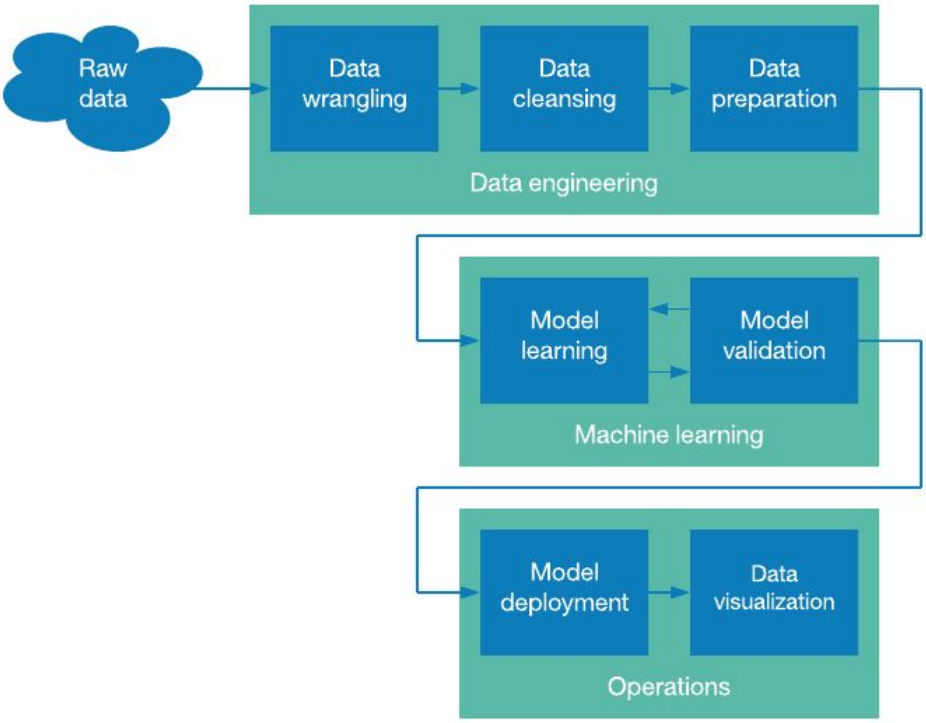
Remember that with our courses, you're not alone! You can contact your mentor to get support on any aspect of your course.

The best way to get help is to login to **www.hyperiondev.com/portal** to start a chat with your mentor. You can also schedule a call or get support via email.

Your mentor is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

PYTHON PACKAGES FOR DATA ENGINEERING

Now that we have a basic understanding of how to obtain data and of some of the techniques we can use to visualize data, we are ready to start data engineering. Remember that data engineering is the first part of the data science pipeline.



(Jones, 2018)

To recap, data engineering involves data wrangling, cleansing and 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 data set. 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.

In the following few tasks, you will learn many techniques needed for data wrangling, cleansing and preparation. It is essential to understand and be able to use Python packages for data science (that you will learn about next) to be able to wrangle and cleanse data.

HOW TO INSTALL PYTHON PACKAGES

In this task, you will be working with three of the most popular Python packages for Data Science, namely: NumPy (documentation **here**), SciPy (documentation **here**) and Pandas (documentation **here**). You will learn more about each of these packages in the remainder of this task. Before getting into the theory though, follow the instructions in this section to install the Python packages you will be using.

The installation instructions differ slightly depending on the operating system you are using. Please follow the relevant instructions below.

HOW TO INSTALL PYTHON PACKAGES ON WINDOWS OS

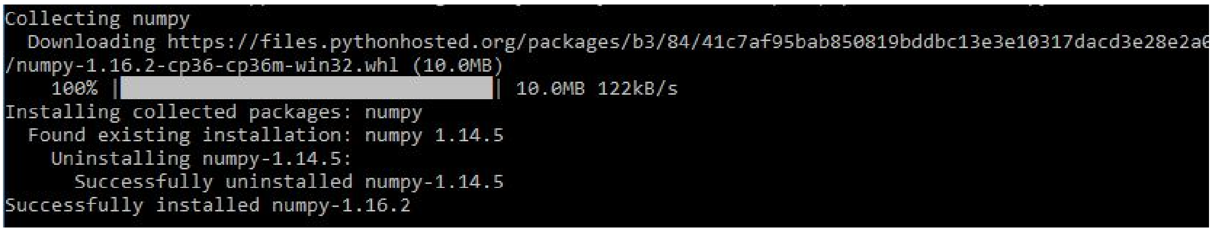
Prerequisites: Python needs to be installed on your computer.

- Open the command prompt (if you are not sure how to use the command line interface, please consult the additional reading for this task).
- Change directory to the 'script' directory in your Python programs folder. See an example of this path below:

  ```
  $ cd C:\Users\HyperionDev\AppData\Local\Programs\Python\Python36-32\Scripts
  ```
- From there, you use pip3.6 to install numpy as shown below. For more information about using pip, see here.

  ```
  $ pip install -U numpy
  ```

  Be patient - this may take a while. Once numpy is installed, you should see output similar to what is shown below in the command line interface:

  ```
  Collecting numpy
    Downloading https://files.pythonhosted.org/packages/b3/84/41c7af95bab850819bddbc13e3e10317dacd3e28e2a
  /numpy-1.16.2-cp36-cp36m-win32.whl (10.0MB)
      100% |████████████████████████████████| 10.0MB 122kB/s
  Installing collected packages: numpy
    Found existing installation: numpy 1.14.5
      Uninstalling numpy-1.14.5:
        Successfully uninstalled numpy-1.14.5
  Successfully installed numpy-1.16.2
  ```
- Now install SciPy and Pandas.

  ```
  $ pip install -U scipy
  $ pip install -U pandas
  ```
- Check that everything is working. You can try the following:
  - Open your command prompt and type 'python' without the quotes. Hit enter.

  - If this doesn't work, it means that you haven't set up Python on the command line correctly. See here for instructions to make sure that Python is in your path if you are using Windows. If you are still having problems, leave a comment in comments.txt or email us ASAP.
  - Now type 'import numpy'. If nothing happens and your cursor goes to the next line - Hooray! If you get any type of error, please make sure that you have correctly followed the three steps listed above.

Note: Please contact your mentor ASAP if you can't get the 'import numpy' statement to work!

A note from our coding mentor
Melanie

*Now that you have pip installed, you can also 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! You can find many more Python libraries and packages **here**.*

HOW TO INSTALL PYTHON LIBRARIES ON LINUX OS

- Let's start by making sure we have an updated system:
  Open the terminal (if you are not sure how to use the command line interface, please consult the additional reading for this task).

```
> sudo apt update
> sudo apt upgrade
```

- Now, let's install NumPy, SciPy and Pandas:

```
> sudo apt install python3-numpy python3-scipy python3-pandas
```

If you did not get any errors, then you have successfully installed the NumPy package and you are ready to learn using NumPy. If you have problems completing the installation, contact your mentor to assist you.

NUMPY

NumPy stands for Numeric Python. It is an open-source extension module for Python and hence uses Python syntax. NumPy is considered to be the 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.

In this task, you will learn how to use the NumPy package to perform basic routines for manipulating large arrays and matrices of numeric data.

Importing NumPy Package

To use NumPy, it has to be imported like other Python modules. There are three ways for this to be achieved. The examples in this task use all three ways of importing NumPy so that you get the hang of it. Feel free to choose the one you prefer.

1. The standard approach uses a simple `import numpy` statement. This will create a reference to the NumPy module in the current namespace. To access NumPy functions using this approach, use `numpy.X,` where X is a NumPy function. For example:

```
import numpy #imports numpy into the current script
f_values = [22.5, 24.9, 27.8, 27.1] #defines a list of floating-point
numbers
numpy_values = numpy.array(f_values) #converts f_values into and array
print(numpy_values) #will output [ 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. Afterwards, when you need to invoke something from NumPy, you can use `nickname.X` instead of `numpy.X`. This is the most commonly used option. For example:

```
import numpy as np
f_values = [22.5, 24.9, 27.8, 27.1]
numpy_values = np.array(f_values)
print(numpy_values) #will output [ 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 every method/function from NumPy.

```
from numpy import *
f_values = [22.5, 24.9, 27.8, 27.1]
numpy_values = array(f_values)
print(numpy_values) #will output [ 22.5 24.9 27.8 27.1]
```

The asterisk (*) implies "everything", so this will import 'everything' from NumPy.

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

```
import numpy as np
f_values = [22.5, 24.9, 27.8, 27.1]
numpy_values = np.array(f_values)
#suppose we want to convert 'degrees' to 'fahrenheit'
#using numpy scalar multiplication
print(numpy_values * 9 / 5 + 32) #will output [ 72.5 76.82 82.04 80.78]

#using core Python
f_values = [x * 9 / 5 + 32 for x in f_values] #iterates through all items in
f_values and multiplies it by 9, divides by 5, adds 32
print(f_values) #will output [ 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!

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).

A note from our coding mentor

Jared

Did you know that Python wasn't originally built for Data Science? And yet today it's one of the best languages for statistics, machine learning, and predictive analytics, as well as simple data analytics tasks. How come? It's an open-source language, and data professionals started creating tools for it to complete data tasks more efficiently.

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 is built on NumPy and that makes it easy to use in NumPy-centric applications, such as data structures with labelled axes. Pandas also provides for explicit data alignment (we'll read more about this later), which prevents common errors that result from misaligned data coming in from different sources. All this makes pandas arguably the best tool for doing data wrangling.

We import pandas 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.).

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

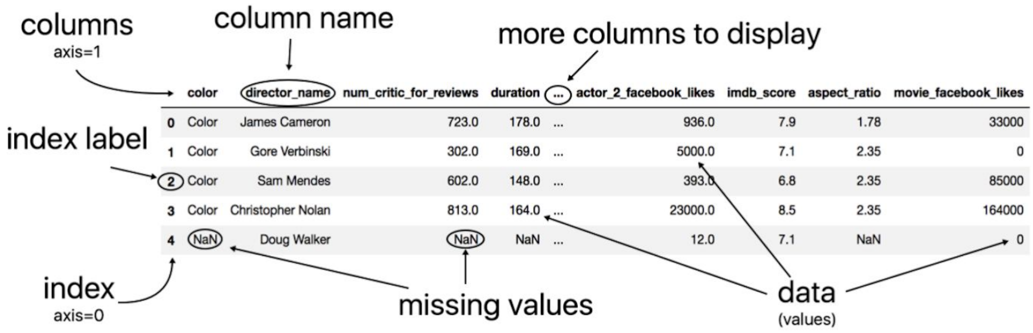
```
import pandas as pd
import numpy as np

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

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

DataFrame

The **pandas library documentation** defines a DataFrame as a “two-dimensional, size-mutable, with labelled rows and columns”.



	color	director_name	num_critic_for_reviews	duration	...	actor_2_facebook_likes	imdb_score	aspect_ratio	movie_facebook_likes
0	Color	James Cameron	723.0	178.0	...	936.0	7.9	1.78	33000
1	Color	Gore Verbinski	302.0	169.0	...	5000.0	7.1	2.35	0
2	Color	Sam Mendes	602.0	148.0	...	393.0	6.8	2.35	85000
3	Color	Christopher Nolan	813.0	164.0	...	23000.0	8.5	2.35	164000
4	NaN	Doug Walker	NaN	NaN	...	12.0	7.1	NaN	0

Anatomy of a DataFrame

Image source: (Petrou, 2017)

In simple terms, 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 data set, DataFrames avoid having missing values, and there are no gaps and empty values between rows or columns.”

You can create a Dataframe by passing a NumPy array, with a datetime index and labeled columns:

```
dates = pd.date_range('20130101', periods=6) # grenerates 6 different days

df = pd.DataFrame(np.random.randn(6,4), index=dates, columns=list('ABCD'))

Output of df:
                   A         B         C         D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
#src of code: (MeteoThink.org, 2018)
```

Or, create a DataFrame by passing a dict of objects that can be converted to series-like:

```
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' })

Output:
    A          B      C      D     E       F
0   1.0   2013-01-02  1.0     3    test    foo
1   1.0   2013-01-02  1.0     3    train   foo
2   1.0   2013-01-02  1.0     3    test    foo
3   1.0   2013-01-02  1.0     3    train   foo
Df2.dtypes           #to get the types for the columns
#src of code: (MeteoThink.org, 2018)
```

Here is how to view the top and bottom rows of the frame:

```
df.head()
Output:
            A        B        C        D
2013-01-01  0.469112 -0.282863 -1.509059 -1.135632
```

```
2013-01-02  1.212112 -0.173215  0.119209 -1.044236
2013-01-03 -0.861849 -2.104569 -0.494929  1.071804
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401

df.tail(3)
Output:
                   A         B         C         D
2013-01-04  0.721555 -0.706771 -1.039575  0.271860
2013-01-05 -0.424972  0.567020  0.276232 -1.087401
2013-01-06 -0.673690  0.113648 -1.478427  0.524988
df.index            # to get the indexes of your data
df.columns        # to get the column names of the data
df.values           # to get the values of the data without the column rows
df.describe()    # shows a quick statistic summary of your data
#src of code: (MeteoThink.org, 2018)
```

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 **example2.py** that accompanies this task to see how the statement above is used to read data from a file called **credit.csv**.

In this task, you have been given a broad overview of NumPy, SciPy and pandas. In the next task, you will learn to work with these packages in more detail.

A note from our coding mentor

Ridhaa

Additional reading*: There is a lot to learn about working with Numpy and Pandas! This task has only scratched the surface. If you would like more information about working with these libraries, please consult the book titled, “**Python Data Science Handbook**” by Jake VanderPlas (Chapters 2 and 3) for more information.*
*You should also consult the official documentation for **Numpy** and **Pandas** as needed.*

Instructions

First read all the example files that accompany this task. Open them using IDLE.

- The example files should help you understand some simple Python.
- You may run the example files to see the output. Feel free to write and run your own example code before doing the Task to become more comfortable with Python.

Compulsory Task 1

Follow these steps:

- Create a new program named **compTask1.py** and add comments and code to answers to 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. (Help **here**)
    - What is the difference between `a = np.array([0, 0, 0])` and `a = np.array([[0, 0, 0]])`?
    - A 3 by 4 by 4 is created with `arr = np.linspace(1, 48, 48).reshape(3, 4, 4)`. Index or slice this array to obtain the following (help **here**, **here** and **here**):
        - 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: use flatten (help **here**) and reshape.

Compulsory Task 2

- Create a new program named **compTask2.py** that will:

- Read in the file **credit.csv** in pandas and name the DataFrame as credit.
- Print the first 10 rows of the DataFrame.
- Print the Age and Education columns of the DataFrames.
- Select users who are above the age of 30.

Completed the task(s)?

Ask your mentor to review your work!

Review work

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** 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://www.scipy.org/scipylib/index.html**