Different Ways of Loading Data using Python

BEGINNER DEEP LEARNING PROGRAMMING PYTHON

This article was published as a part of the Data Science Blogathon.

Introduction

When we started our data science journey the very first library that got our way was **pandas** and the very first function was **read_csv(). But do we have only a CSV file to read? No right!** This article will let you know about other methods one can use to read and access the dataset.

In this article, we will work with **different ways of loading data using python**. The motivation behind writing this blog is when I searched about the same and got minimal sites that are shifting their focus from **CSV format** to any other available method. However, there is no doubt about the fact that **PD.read_csv()** is one of the best ways of reading datasets using python (**pandas-** particularly). While working on a real-time project, we should know different methods of **loading and accessing the dataset** as no one knows what turns out to be handy at what time.

So, let's first see what kind of methods we are gonna be working then straightaway implement it using **Python.**

Source: Google Digital Garage

Loading Data in Python Using 5 different methods

1. **Manually loading a file:** This is the first, most **popular,** and **least recommended way to load data as it requires many code parts to** read one tuple from the DataFrame. This way comes into the picture when the dataset **doesn't have any particular pattern to identify** or a **specific pattern.**

2. I am using **np. load txt:** One of the **NumPy methods** for loading different types of data though it is only supported when we have data in a specific format, i.e., **pattern recognizable,** unlike the manual way of reading the dataset.

3. Using **np.GenFromTxt:** This is another NumPy way to read the data, but this time it is much better than the **np. load txt()** method recognizes the column header's presence on its own, which the previous one cannot follow. Along with that, it can also detect the **right data type** for each column.

4. Using **PD.read_csv:** Here is the most recommended and widely used method for **reading, writing, and manipulating** the dataset. It only deals with **CSV formatted** data, but the support of various parameters makes **it a gold mine for data analysts** to work with different sorts of data (they should have a specific format).

5. Using **pickle:** Last but not least, we will also use **a pickle** to read the dataset present in the **binary format**. So far, we chose pickle only to save the model**,** but the capabilities of this module are much more than that which we will explore in the future article.

Importing the Required Python Libraries

The first thing for using the python supported libraries we first need to import them to establish the environment and configuration. Here are the following modules that we will import.

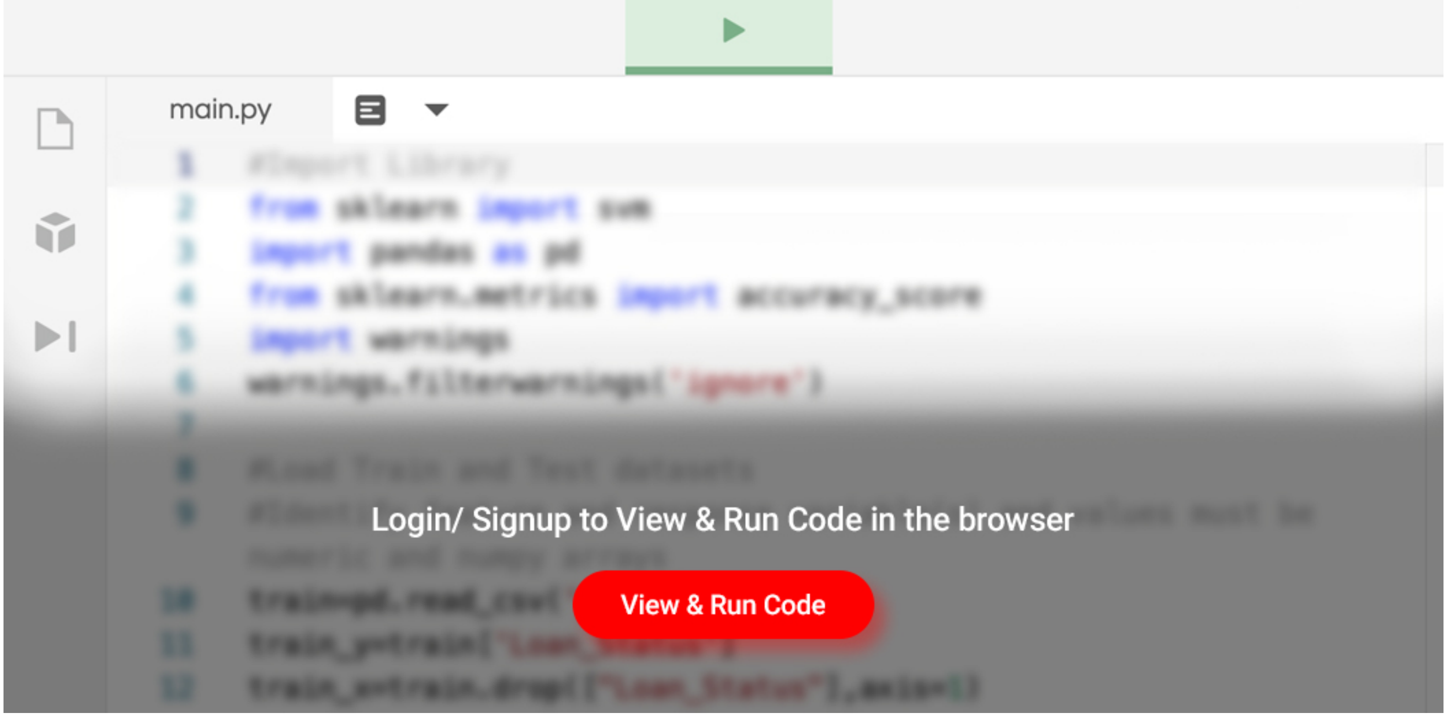
- **Numpy:** Numpy is mainly used to perform mathematical calculations though here we will use it to access the methods for our needs. They are **np. load txt()** and **np. GenFromTxt()**.
- **Pandas:** For **converting** the unsorted data into the **correctly formatted** [dataset](), we can later manipulate and draw some insights.
- **Pickle:** We need to import the same independently for using the pickle module.

```
import numpy as np import pickle import pandas as pd file_dir = "load_dataset_blog.csv"
```

So we have imported the modules we might need and stored the dataset's path in a variable so that we don't have to write it again and again.

Manually Reading the Dataset

Here comes the first method, where we will use the **open** and **readLines** method by looping through each line. For that reason, it is the least recommended and exceptionally used method as it is not flexible and feasible enough to make our task easier instead of a bit hectic.



```
#Import Library
from sklearn import svm
import pandas as pd
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')

#Load Train and Test datasets
#Identif[illegible] values must be
numeric and numpy arrays
train=pd.read_csv([illegible]
train_y=train['Loan_Status']
train_x=train.drop(["Loan_Status"],axis=1)
```

Code breakdown:

1. Firstly, we are using an instance **with** an **open** function to read the dataset. Then to access each tuple, we have to loop through the complete dataset till the end of the file (**readLines()**).
2. Then, one will notice that we have also used the **replace** function as after every line, there is the line break which is not needed while building up the DataFrame.
3. Then, the column names and data (tuples) are separated. Note that for storing data values, we are using the **list comprehension** method. At last, using pandas' **DataFrame** function the unstructured data is converted into a pandas dataset.

np.loadtxt

This method is widely used **for simple data arrays** requiring very minimal formatting, i.e., for simple values where such changes are unnecessary. Here we also use some mandatory parameters that will be discussed in no time. We can also use the **np. save_txt** function to save the data read/loaded by **np. load_txt.**

```
d1 = np.loadtxt(filename, skiprows=1, delimiter=",") print(d1.dtype) print(d1[:5, :])
```

Output:

Inference: np. load txt() method is used with relevant parameters like **filename** to get the path of the dataset, and **skip rows**, which is responsible to decide whether the first row (column headers) should be skipped or not. The **delimiter** specifies how the values are separated as in our case; it's a comma (",").

For printing each row (5 here), we use the **slicing** concept of python list data type. Note that we cannot see the column header in output because **skip rows are set to 1.**

np. GenFromTxt

This is another function supported by NumPy that is more flexible than the **np. Load txt** function as **np. GenFromTxt** has better parsing functionality in supporting different **types of data**, **named columns**.

```
d2 = np.genfromtxt(filename, delimiter=",", names=True, dtype=None) print(d2.dtype) print(d2[:5])
```

Output:

Inference: Starting with focussing on the parameter where filename is used to access the path of the dataset, then delimiter is tagged as comma as we are working with CSV file, names are set to be True so that columns are visible. Lastly, type is set to None so that NumPy will autodetect the column type ((‘E’, ‘**<i8**’) – integer detected)

pandas.read_csv

By far the best and most flexible CSV/txt file reader. Highly recommended. If you don’t believe me, just look at the obscene number of arguments you can parse to read_csv in the documentation.

```
d3 = pd.read_csv(filename) print(d3.dtypes) d3.head()
```

Output:

Inference: There is not much to discuss here, as reading the CSV file from pandas is something from where we started our data science journey. Note that just like the previous function it can also detect the real data type of each column (**E – int64**).

Pickle

When your data or object is not a nice 2D array and harder to save as something human readable. Note that if you just have a 3D, 4D… ND array of all the same type, you can also use **np. save,** which will save an arbitrary NumPy array in binary format. Super quick to save, super quick to load in, and small file size.

Pickle is for everything more complicated. You can save dictionaries, arrays, and even objects.

```
with open("load_pickle.pickle", "rb") as f: d4 = pickle.load(f) print(d4.dtypes) d4.head()
```

Output:

Inference: The last method for loading/reading the data is Pickle, we have often used this for only saving the model, but here, we can see that it can also read the same in the **read mode** (note that it is serialized in the binary format). For loading the pickle file, we use the **load** function.

Conclusion

Here we are in the last part of the article; In this article, we learned the different ways the data can be read using **numerous functions available with Python**. While practically looking at the differences, we also learned the limitations of each way and **when to use what**.

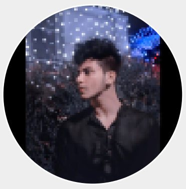
1. First, we started with manually reading the file and concluded that it is highly not recommended to use this method as it is **time-consuming**.
2. Then we discussed two methods that NumPy supports (**np. load txt** and **np. GenFromTxt**), where the second one seems **more optimal** as it was even able to detect the right data type for each column.
3. At last, we parsed through the last two ways (**read_csv** and **pickle**), where we got an interesting insight- **pickle is just not for saving models** but also for training/testing datasets, dictionaries, objects, and what-not.

Here's the repo link to this article. I hope you liked my article on **Different ways of loading data using Python**. If you have any opinions or questions, comment below.

Connect with me on LinkedIn for further discussion on Python for Data Science or otherwise.

The media shown in this article is not owned by Analytics Vidhya and is used at the Author's discretion.

Article Url - https://www.analyticsvidhya.com/blog/2022/08/different-ways-of-loading-data-using-python/



Aman Preet Gulati