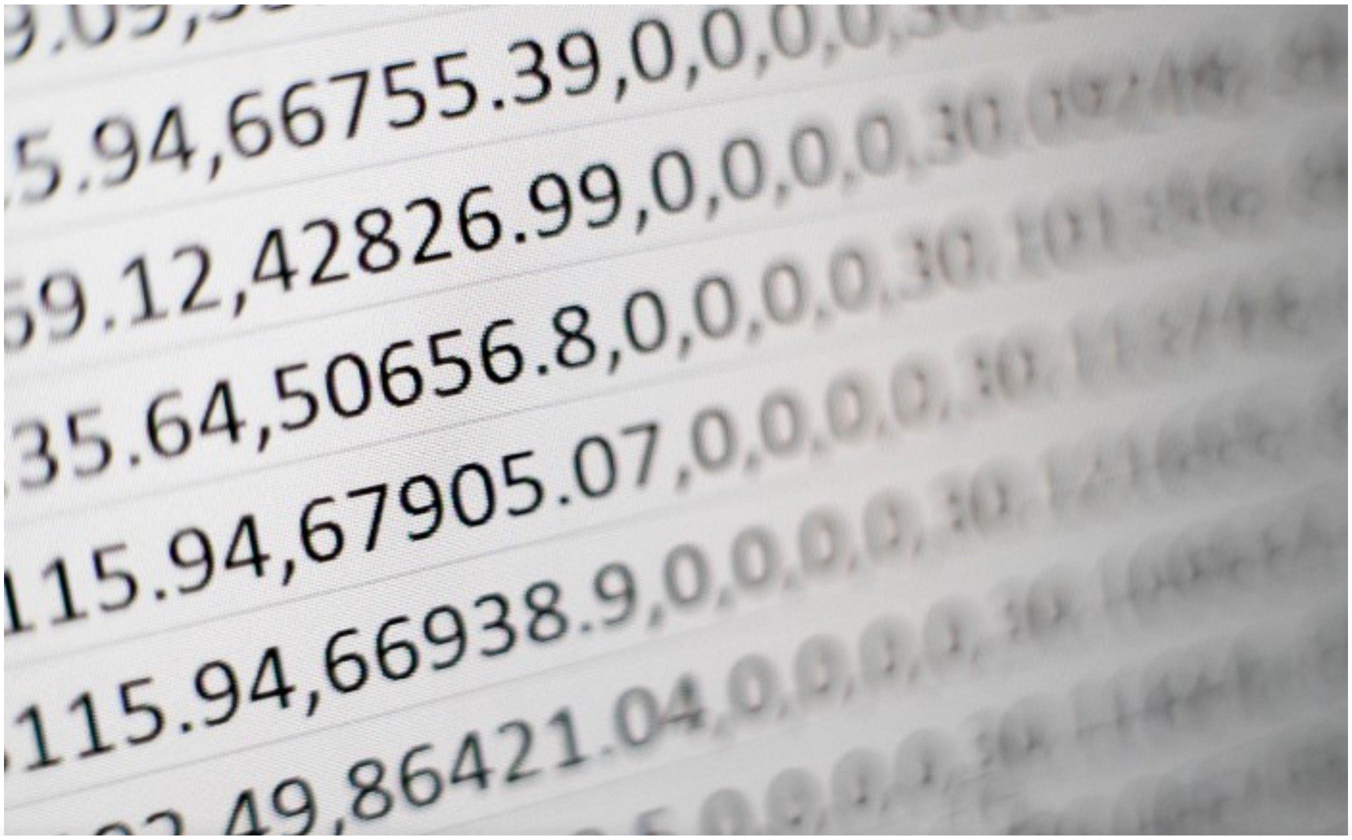
Published in Towards Data Science

Georgia Deaconu Follow
Jan 1 · 3 min read · Listen

3 ways to deal with large datasets in Python

As a data scientist, I find myself more and more having to deal with "*big data*". What I abusively call *big data* corresponds to datasets that, while not really that large, are large enough to make my computer struggle to process them and really slow everything down.



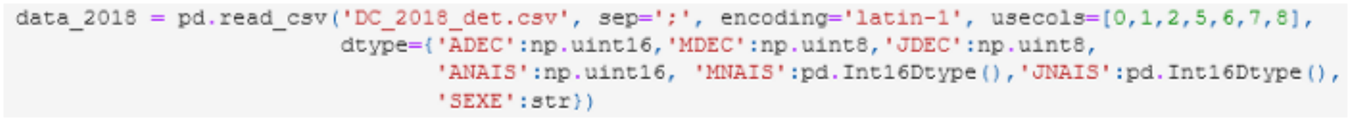
Get unlimited access   Open in app

This problem is not new and as with everything, there is no one size fits it all magic formula. The best method will depend on your data and the purpose of your application. However, the most popular solutions usually fall in one of the categories described below.

1. Reduce memory usage by optimizing data types

When using Pandas to load data from a file, it will automatically infer data types unless told otherwise. Most of the times this will work fine but the inferred type is not necessarily optimized. Moreover, if a numerical column contains missing values than the inferred type will automatically be float.

I have used this method recently for an analysis where I was dealing mostly with integer types representing years, months or days :



```
data_2018 = pd.read_csv('DC_2018_det.csv', sep=';', encoding='latin-1', usecols=[0,1,2,5,6,7,8],
                        dtype={'ADEC':np.uint16,'MDEC':np.uint8,'JDEC':np.uint8,
                               'ANAIS':np.uint16, 'MNAIS':pd.Int16Dtype(),'JNAIS':pd.Int16Dtype(),
                               'SEXE':str})
```

Use Pandas to load file and specify dtypes (Image by Author)

For that particular case, specifying the data types led to an important reduction of the used memory. Please notice that in the example above I used the pandas type pandas.Int16Dtype to force a column containing missing values to be of type int. This is achieved internally by using pandas.NA rather than numpy.nan for the missing values

Optimizing dtypes when handling large datasets requires to already have some prior knowledge of the data you are dealing with. It might not be useful in a purely exploratory phase of an unknown dataset.

2. Split data into chunks

When data is too large to fit into memory, you can use Pandas' *chunksize* option to split the data into chunks instead of dealing with one big block. Using this option

Here is an example of using this option to go through the Yelp reviews dataset, extract the minimum and maximum review date for each chunk, and then rebuild the complete time span for the reviews :

```
reader = pd.read_json(reviews_path, lines=True, orient = "records", chunksize = 100000)

# go through the chunks and extract min/max date
date_limits = []
for chunk in reader:

    date_limits.append(max(chunk.date))
    date_limits.append(min(chunk.date))

print("Reviews span from {} to {}".format(min(date_limits).strftime('%d-%m-%Y'),
                                          max(date_limits).strftime('%d-%m-%Y')))
```

gistfile1.txt hosted with ❤ by **GitHub** **view raw**

Chunking can be used from initial exploratory analysis to model training and requires very little extra setup.

3. Take advantage of lazy evaluation

Lazy evaluation refers to the strategy which delays evaluation of an expression until the value is actually needed. Lazy evaluation is an important concept (used especially in functional programming), and if you want to read more about its different usages in Python you can start here.

Lazy evaluation is the basis on which distributed computation frameworks such as Spark or Dask are built. Although they were designed to work on clusters you can still take advantage of them to handle large datasets on your personal computer.

The main difference with respect to Pandas is that they do not load the data directly in memory. Instead what happens during the read command is that they scan the data, infer dtypes and split it into partitions (so far nothing new). Computational another post so I will not repeat any of it here. Dask is also quite popular and examples are not hard to find (for a comparison between the two you can start here). Dask syntax mimics Pandas' so it will look very familiar, however, it is limited to Python usage, while Spark will also work in Java or Scala.

Other libraries such as Vaex or Modin offer similar capabilities but I have not used them personally.