**Import data**

In this course, you'll learn how to import data from a large variety of import data sources, for example, (i) flat files such as dot txts and dot csvs; (ii) files native to other software such as Excel spreadsheets, Stata, SAS and MATLAB files;

(iii) relational databases such as SQLite & PostgreSQL. We’ll cover all of these topics in this course.

**Plain text files**

First off, we're going to learn how to import basic text files, which we can broadly classify into 2 types of files - those containing plain text, such as the opening of Mark Twain's novel The Adventures of Huckleberry Finn, which you can see here,

**Table data**

and those containing records, that is, table data, such as titanic dot csv, in which each

1. 1 Source: Kaggle

row is a unique passenger onboard and each column is a characteristic or feature, such as gender, cabin and 'survived or not'. The latter is known as a flat file and we'll come back to these in a minute.

**Reading a text file**

In this section, we'll figure out how to read lines from a plain text file: So let's do it! To check out any plain text file, you can use Python’s basic open function to open a connection to the file. To do so, you assign the filename to a variable as a string, pass the filename to the function open and also pass it the argument mode equals 'r', which makes sure that we can only read it (we wouldn't want to accidentally write to it!), assign the text from the file to a variable text by applying the method read to the connection to the file. After you do this, make sure that you close the connection to the file using the command file dot close. It’s always best practice to clean while cooking!



**Printing a text file**

You can then print the file to console and check it out using the command print(text). A brief side note:

**Writing to a file**

if you wanted to open a file in order to write to it, you would pass it the argument mode equals 'w'. We won't use that in this course as this is course on Importing Data but it is good to know. You can avoid having to close the connection to the file by



**Context manager with**

using a with statement. This allows you to create a context in which you can execute commands with the file open. Once out of this clause/context, the file is no longer open and, for this reason, with is called a Context Manager. What you're doing here is called 'binding' a variable in the context manager construct; while still within this construct, the variable file will be bound to open(filename, 'r'). It is best practice to use the with statement as you never have to concern yourself with closing the files again.



**.**

**The importance of flat files in data science**

**Flat files**

we're going to look at flat files, such as 'titanic dot csv',

**Flat files**

in which each row is a unique passenger onboard and each column is a feature of attribute, such as gender, cabin and 'survived or not'. It is essential for any budding data scientist to know precisely what the term flat file means.

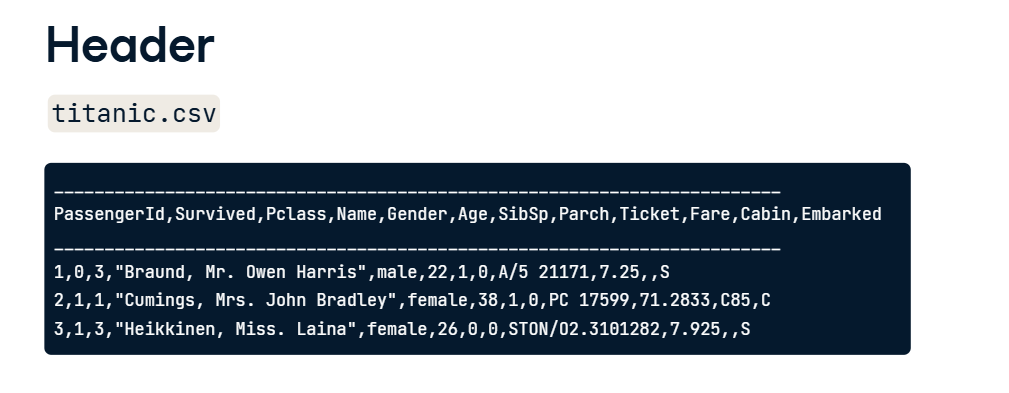
**Flat files**

Flat files are basic text files containing records, that is, table data, without structured relationships. This is in contrast to a relational database, for example, in which columns of distinct tables can be related. We'll get to these later. To be even more precise, flat files consist of records, where by a record we mean a row of fields or attributes, each of which contains at most one item of information. In the flat file 'titanic dot csv', each

row or record is a unique passenger onboard and each column is a feature or attribute, such as name, gender and cabin.

**Header**

It is also essential to note that a flat file can have a header, such as in 'titanic dot csv', which is a row that occurs as the first row and describes the contents of the data columns or states what the corresponding attributes or features in each column are. It will be important to know whether or not your file has a header as it may alter your data import. The reason that flat files are so important in data science is that we data scientists really honestly like to think in records or rows of attributes.

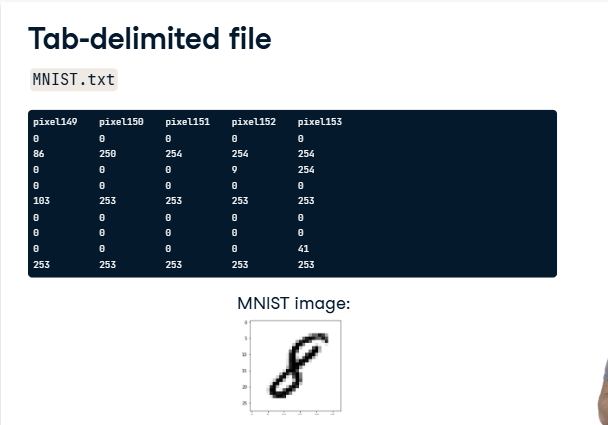


**File extension**

Now you may have noticed that the file extension was dot csv. You may be wondering what this is? Well, CSV is an acronym for comma separated value and it means exactly what it says. The values in each row are separated by commas. Another common extension for a flat file is dot txt, which means a text file. Values in flat files can be separated by characters or sequences of characters other than commas, such as a tab, and the character or characters in question is called a delimiter.

**Tab-delimited file**

See here an example of a tab-delimited file. The data consists of the famous MNIST digit recognition images, where each row contains the pixel values of a given image. Note that all fields in the MNIST data are numeric, while the 'titanic dot csv' also contained strings.



**How do you import flat files?**

How do we import such files? If they consist entirely of numbers and we want to store them as a numpy array, we could use numpy. If, instead, we want to store the data in a dataframe, we could use pandas. Most of the time, you will use one of these options. In the rest of this Chapter, you'll learn how to import flat files that contain only numerical data, such as the MNIST data, and import flat files that contain both numerical data and strings, such as 'titanic dot csv'.

**Importing flat files using NumPy**

Okay so you now know how to use Python’s built-in open function to open text files. What if you now want to import a flat file and assign it to a variable? If all the data are numerical, you can use the package numpy to import the data as a numpy array. Why would we want to do this?

**Why NumPy?**

First off, numpy arrays are the Python standard for storing numerical data. They are efficient, fast and clean.

**Why NumPy?**

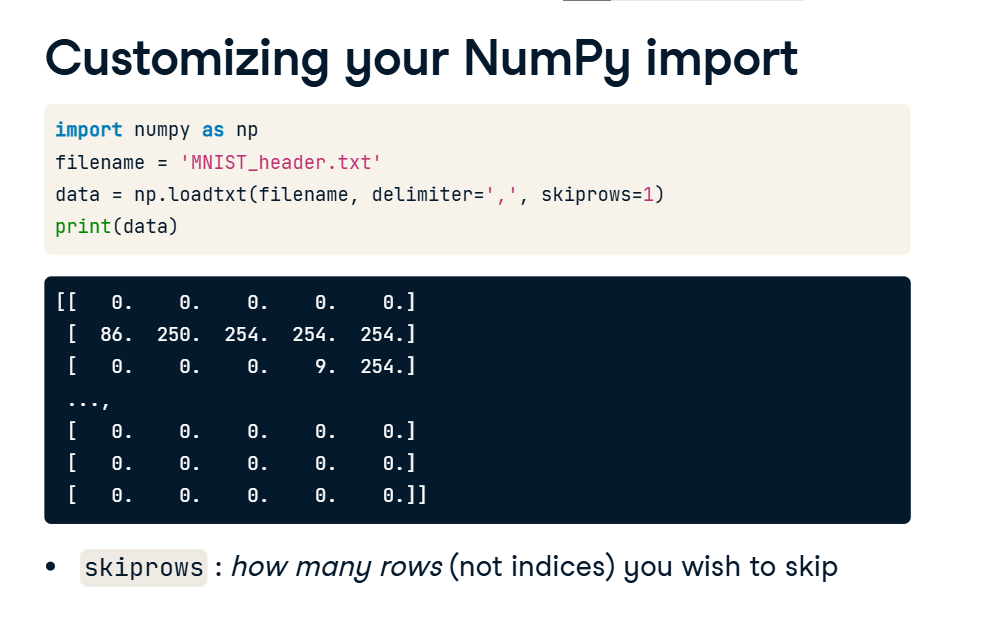
Secondly, numpy arrays are often essential for other packages, such as scikit-learn, a popular Machine Learning package for Python. Numpy itself has a number of built-in functions that make it far easier and more efficient for us to import data as arrays. Enter the NumPy functions loadtxt and genfromtxt.

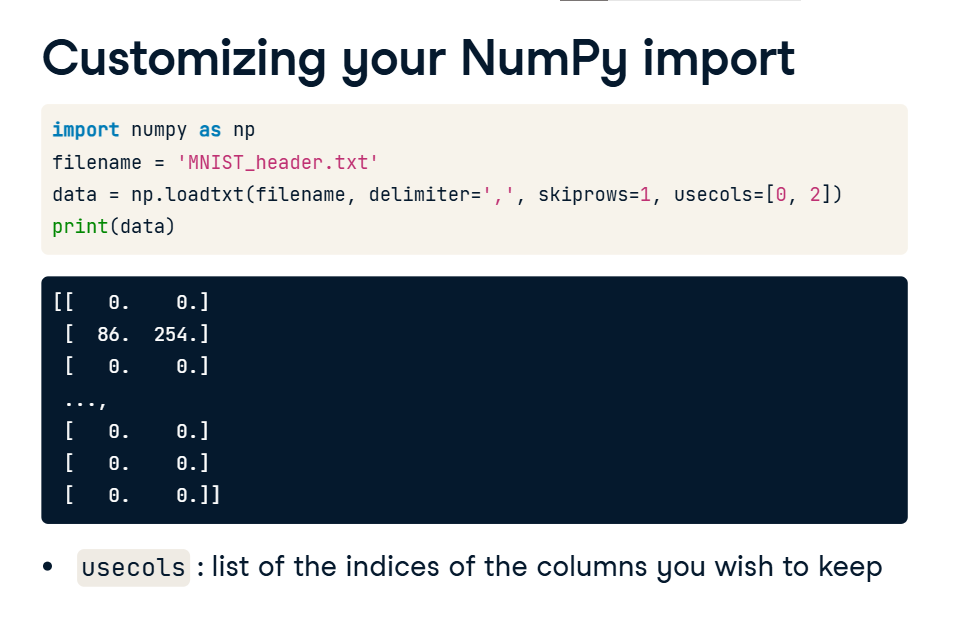
**Importing flat files using NumPy**

To use either of these we first need to import NumPy. We then call loadtxt and pass it the filename as the first argument, along with the delimiter as the 2nd argument. Note that the default delimiter is any white space so we’ll usually need to specify it explicitly.

**Customizing your NumPy import**

There are a number of additional arguments you may wish to specify. If, for example, your data consists of numerics and your header has strings in it, such as in the MNIST digits data, you will want to skip the first row by calling loadtxt with the argument skiprows equals 1; if you want only the 1st and 3rd columns of the data, you’ll want to set usecols equals the list containing ints 0 and 2.



****

**Customizing your NumPy import**

You can also import different datatypes into NumPy arrays: for example, setting the argument dtype equals 'str' will ensure that all entries are imported as strings. loadtxt is great for basic cases, but tends to break down when we have



**Mixed datatypes**

mixed datatypes, for example,

1. 1 Source: Kaggle

columns consisting of floats AND columns consisting of strings, such as we saw in the Titanic dataset.

**1. Importing flat files using pandas**

00:00 - 00:17

Congrats! You're now able to import a bunch of different types of flat files into Python as NumPy arrays. Although arrays are incredibly powerful and serve a number of essential purposes, they cannot fulfill one of the most basic needs of a Data Scientist:

**2. What a data scientist needs**

00:17 - 00:42

to have "[two]-dimensional labeled data structure[s] with columns of potentially different types" that you can easily perform a plethora of Data Sciencey type things on: manipulate, slice, reshape, groupby, join, merge, perform statistics in a missing-value-friendly manner, deal with times series. The need for such a data structure, among other issues,

**3. Pandas and the DataFrame**

00:42 - 00:45

prompted Wes McKinney to develop the

**Pandas and the DataFrame**

pandas library for Python. Nothing speaks to the project of pandas more than the documentation itself:

**Pandas and the DataFrame**

"Python has long been great for data munging and preparation, but less so for data analysis and modeling. pandas helps fill this gap, enabling you to carry out your entire data analysis workflow in Python without having to switch to a more domain specific language like R". The data structure most relevant to the data manipulation and analysis workflow that pandas offers is the DataFrame and it is the Pythonic analogue of R's data frame.

**Pandas and the DataFrame**

As Hadley Wickham tweeted, "A matrix has rows and columns. A data frame has observations and variables."

**Manipulating pandas DataFrames**

Manipulating DataFrames in pandas can be useful in all steps of the data scientific method, from exploratory data analysis to data wrangling, preprocessing, building models and visualization. Here, we will see its great utility in importing flat files, even merely in the way that it deals with missing data, comments along with the many other issues that plague working data scientists. For all of these reasons, it is now standard and best practice in Data Science to use pandas to import flat files as DataFrames. Later in this course, we'll see how many other types of data, whether they're stored in relational databases, hdf5, MATLAB or excel files, can easily be imported as DataFrames.

**Importing using pandas**

To use pandas, you first need to import it. Then, if we wish to import a CSV in the most basic case all we need to do is to call the function read\_csv() and supply it with a single argument, the name of the file. Having assigned the DataFrame to the variable data, we can check the first 5 rows of the DataFrame, including the header, with the command 'data.head'. We can also easily convert the DataFrame to a numpy array. Now it's your turn to play around importing flat files using Python.

