Shane Lynn

Data science, Startups, Analytics, and Data visualisation.

Python Pandas read_csv – Load Data from CSV Files

6 Comments / blog, data science, Pandas, python, Tutorials / By shanelynn

CSV (comma-separated value) files are a common file format for transferring and storing data. The ability to read, manipulate, and write data to and from CSV files using Python is a key skill to master for any data scientist or business analysis. In this post, we'll go over what CSV files are, how to read CSV files into Pandas DataFrames, and how to write DataFrames back to CSV files post analysis.

Pandas is the most popular data manipulation package in Python, and DataFrames are the Pandas data type for storing tabular 2D data.

Load CSV files to Python Pandas

The basic process of loading data from a CSV file into a Pandas DataFrame (with all go well) is achieved using the "read_csv" function in Pandas:

```python
# Load the Pandas libraries with alias 'pd'
import pandas as pd

# Read data from file 'filename.csv'
# (in the same directory that your python process is based)
# Control delimiters, rows, column names with read_csv (see later)
data = pd.read_csv("filename.csv")

# Preview the first 5 lines of the loaded data
data.head()
```

While this code seems simple, an understanding of three fundamental concepts is required to fully grasp and debug the operation of the data loading procedure if you run into issues:

1. **Understanding file extensions and file types** – what do the letters CSV actually mean? What's the difference between a .csv file and a .txt file?
2. **Understanding how data is represented inside CSV files –** if you open a CSV file, what does the data actually look like?
3. **Understanding the Python path and how to reference a file** – what is the absolute and relative path to the file you are loading? What directory are you working in?
4. **CSV data formats and errors** – common errors with the function.

Each of these topics is discussed below, and we finish this tutorial by looking at some more advanced CSV loading mechanisms and giving some broad advantages and disadvantages of the CSV format.

1. File Extensions and File Types

The first step to working with comma-separated-value (CSV) files is understanding the concept of file types and file extensions.

1. Data is stored on your computer in individual "files", or containers, each with a different name.
2. Each file contains data of different types – the internals of a Word document is quite different from the internals of an image.
3. Computers determine how to read files using the "file extension", that is the code that follows the dot (".") in the filename.
4. So, a filename is typically in the form "<random name>.<file extension>". Exar

- **project1.DOCX** – a Microsoft Word file called Project1.
- **shanes_file.TXT** – a simple text file called shanes_file
- **IMG_5673.JPG** – An image file called IMG_5673.
- Other well known file types and extensions include: XLSX: Excel, PDF: Portable Document Format, PNG – images, ZIP – compressed file format, GIF – animation, MPEG – video, MP3 – music etc. See a complete list of extensions here.

5. A CSV file is a file with a ".csv" file extension, e.g. "data.csv", "super_information.csv". The "CSV" in this case lets the computer know that the data contained in the file is in "comma separated value" format, which we'll discuss below.

File extensions are hidden by default on a lot of operating systems. The first step that any self-respecting engineer, software engineer, or data scientist will do on a new computer is to ensure that file extensions are shown in their Explorer (Windows) or Finder (Mac) windows.

Folder with file extensions showing. Before working with CSV files, ensure that you can see your file extensions in your operating system. Different file contents are denoted by the file extension, or letters after the dot, of the file name. e.g. TXT is text, DOCX is Microsoft Word, PNG are images, CSV is comma-separated value data.

To check if file extensions are showing in your system, create a new text document with Notepad (Windows) or TextEdit (Mac) and save it to a folder of your choice. If you can't see the ".txt" extension in your folder when you view it, you will have to change your settings.

- In Microsoft Windows: Open Control Panel > Appearance and Personalization. N
click on **Folder Options** or **File Explorer Option**, as it is now called > View

tab. In this tab, under Advance Settings, you will see the option **Hide extensions for known file types.** Uncheck this option and click on Apply and OK.

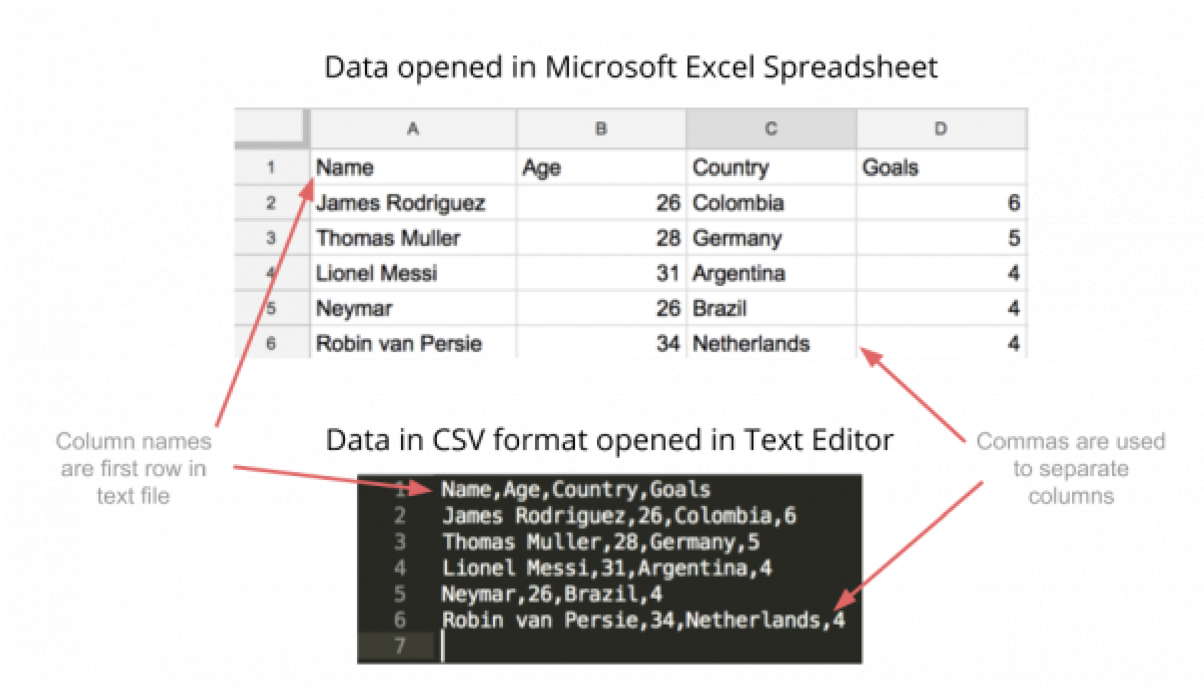
- In Mac OS: Open Finder > In menu, click Finder > Preferences, Click Advanced, Select the checkbox for “Show all filename extensions”.

2. Data Representation in CSV files

A “CSV” file, that is, a file with a “csv” filetype, is a basic text file. Any text editor such as NotePad on windows or TextEdit on Mac, can open a CSV file and show the contents. Sublime Text is a wonderful and multi-functional text editor option for any platform.

CSV is a standard for storing tabular data in text format, where commas are used to separate the different columns, and newlines (carriage return / press enter) used to separate rows. Typically, the first row in a CSV file contains the names of the columns for the data.

And example table data set and the corresponding CSV-format data is shown in the diagram below.


Data opened in Microsoft Excel Spreadsheet
Column names are first row in text file
Data in CSV format opened in Text Editor
Commas are used to separate columns

	A	B	C	D
1	Name	Age	Country	Goals
2	James Rodriguez	26	Colombia	6
3	Thomas Muller	28	Germany	5
4	Lionel Messi	31	Argentina	4
5	Neymar	26	Brazil	4
6	Robin van Persie	34	Netherlands	4

```
Name,Age,Country,Goals
James Rodriguez,26,Colombia,6
Thomas Muller,28,Germany,5
Lionel Messi,31,Argentina,4
Neymar,26,Brazil,4
Robin van Persie,34,Netherlands,4
```

Comma-separated value files, or CSV files, are simple text files where commas and newlines are used to define tabular data in a structured way.

Note that almost any tabular data can be stored in CSV format – the format is popular because of its simplicity and flexibility. You can create a text file in a text editor, save it with a .csv extension, and open that file in Excel or Google Sheets to see the table form.

Other Delimiters / Separators – TSV files

The comma separation scheme is by far the most popular method of storing tabular data in text files.

However, the choice of the ‘,’ comma character to delimiters columns, however, is arbitrary, and can be substituted where needed. Popular alternatives include tab (“\t”) and semi-colon (“;”). Tab-separate files are known as TSV (Tab-Separated Value) files.

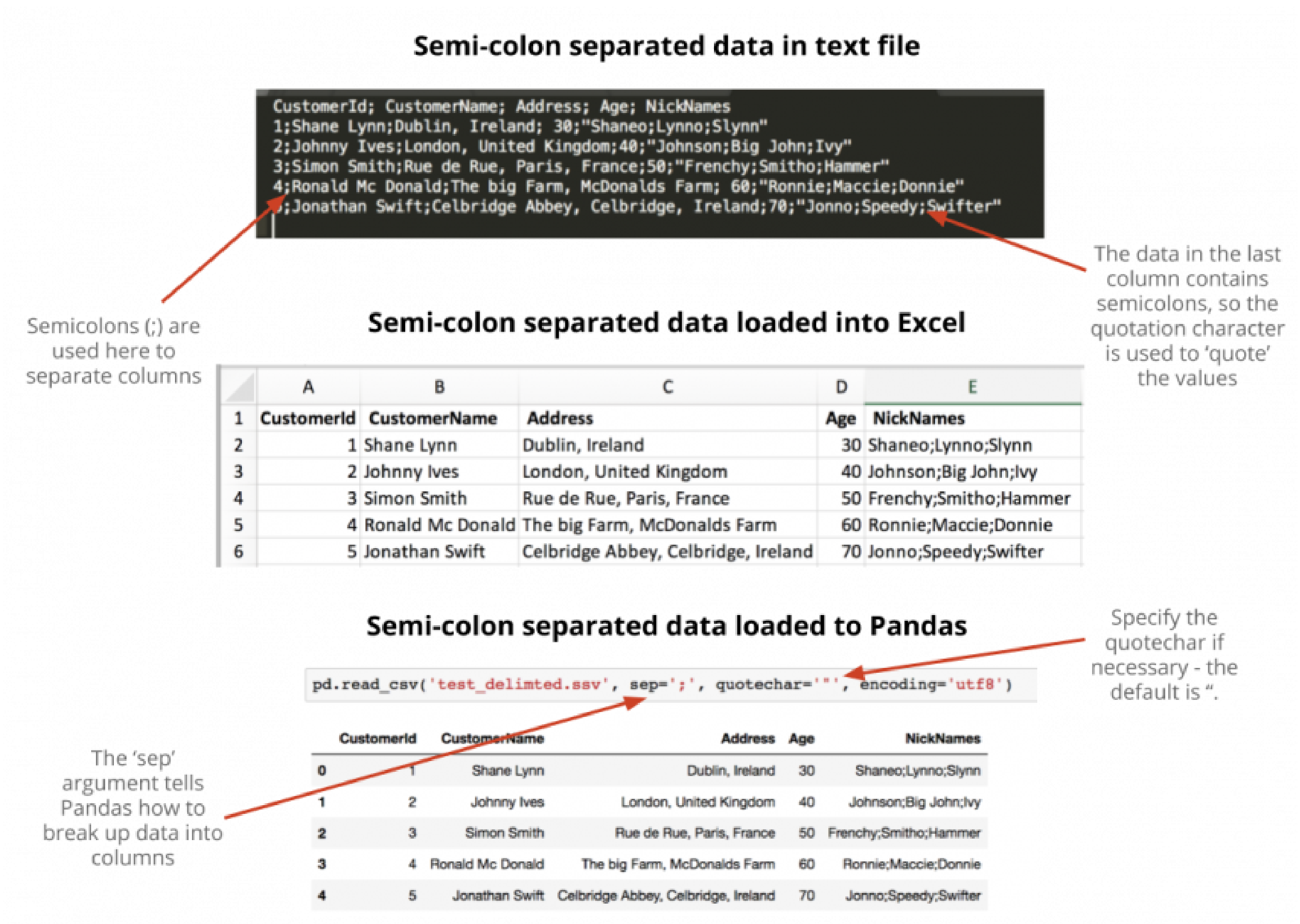
When loading data with Pandas, the read_csv function is used for reading any delimited text file, and by changing the delimiter using the `sep` parameter.

Delimiters in Text Fields – Quotechar

One complication in creating CSV files is if you have commas, semicolons, or tabs actually in one of the text fields that you want to store. In this case, it’s important to use a “quo character” in the CSV file to create these fields.

The quote character can be specified in Pandas.read_csv using the `quotechar` argument. By default (as with many systems), it's set as the standard quotation marks ("). Any commas (or other delimiters as demonstrated below) that occur between two quote characters will be ignored as column separators.

In the example shown, a semicolon-delimited file, with quotation marks as a quotechar is loaded into Pandas, and shown in Excel. The use of the quotechar allows the "NickName" column to contain semicolons without being split into more columns.


Semi-colon separated data in text file
Semicolons (;) are used here to separate columns
The data in the last column contains semicolons, so the quotation character is used to 'quote' the values
Semi-colon separated data loaded into Excel
Semi-colon separated data loaded to Pandas
Specify the quotechar if necessary - the default is ".
The 'sep' argument tells Pandas how to break up data into columns

```
CustomerId; CustomerName; Address; Age; NickNames
1;Shane Lynn;Dublin, Ireland; 30;"Shaneo;Lynno;Slynn"
2;Johnny Ives;London, United Kingdom;40;"Johnson;Big John;Ivy"
3;Simon Smith;Rue de Rue, Paris, France;50;"Frenchy;Smitho;Hammer"
4;Ronald Mc Donald;The big Farm, McDonalds Farm; 60;"Ronnie;Maccie;Donnie"
5;Jonathan Swift;Celbridge Abbey, Celbridge, Ireland;70;"Jonno;Speedy;Swifter"
```

	A	B	C	D	E
1	CustomerId	CustomerName	Address	Age	NickNames
2	1	Shane Lynn	Dublin, Ireland	30	Shaneo;Lynno;Slynn
3	2	Johnny Ives	London, United Kingdom	40	Johnson;Big John;Ivy
4	3	Simon Smith	Rue de Rue, Paris, France	50	Frenchy;Smitho;Hammer
5	4	Ronald Mc Donald	The big Farm, McDonalds Farm	60	Ronnie;Maccie;Donnie
6	5	Jonathan Swift	Celbridge Abbey, Celbridge, Ireland	70	Jonno;Speedy;Swifter

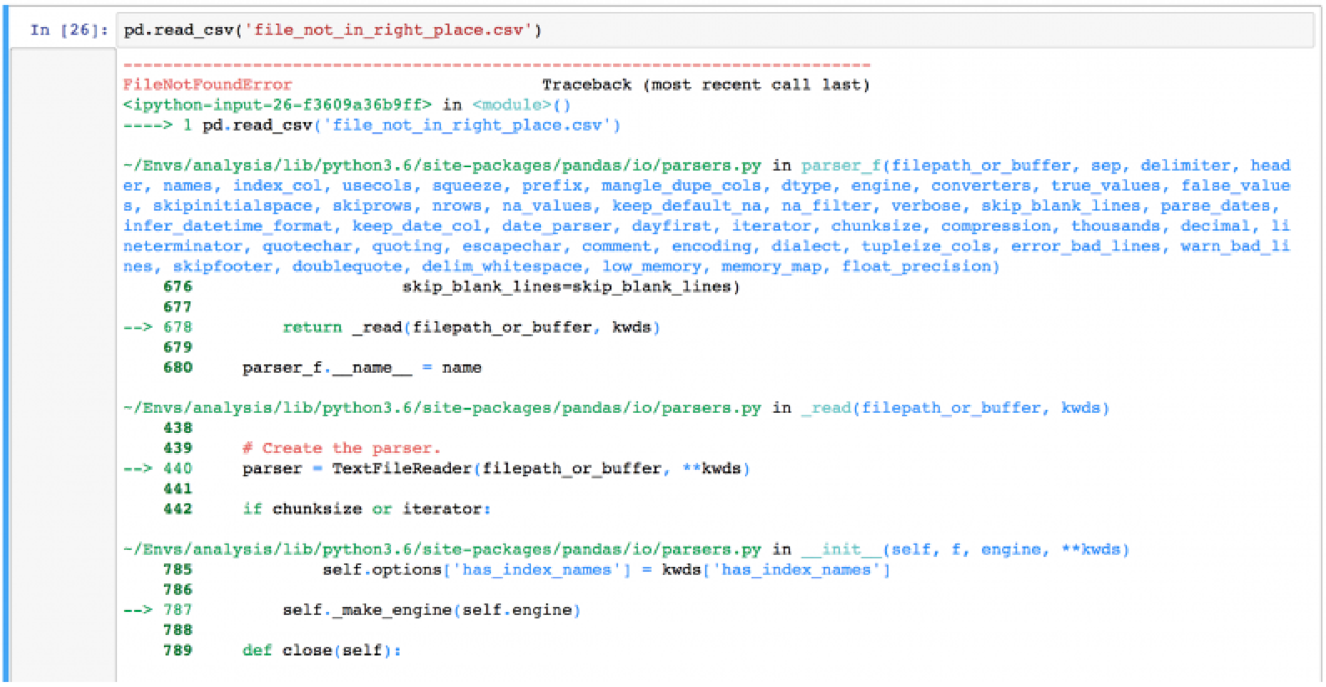
```
pd.read_csv('test_delimted.ssv', sep=';', quotechar='"', encoding='utf8')
```

	CustomerId	CustomerName	Address	Age	NickNames
0	1	Shane Lynn	Dublin, Ireland	30	Shaneo;Lynno;Slynn
1	2	Johnny Ives	London, United Kingdom	40	Johnson;Big John;Ivy
2	3	Simon Smith	Rue de Rue, Paris, France	50	Frenchy;Smitho;Hammer
3	4	Ronald Mc Donald	The big Farm, McDonalds Farm	60	Ronnie;Maccie;Donnie
4	5	Jonathan Swift	Celbridge Abbey, Celbridge, Ireland	70	Jonno;Speedy;Swifter

Other than commas in CSV files, Tab-separated and Semicolon-separated data is popular also. Quote characters are used if the data in a column may contain the separating character. In this case, the 'NickName' column contains semicolon characters, and so this column is "quoted". Specify the separator and quote character in pandas.read_csv

3. Python – Paths, Folders, Files

When you specify a filename to Pandas.read_csv, Python will look in your "**current working directory**". Your working directory is typically the directory that you starte your Python process or Jupyter notebook from.



```
In [26]: pd.read_csv('file_not_in_right_place.csv')

---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
<ipython-input-26-f3609a36b9ff> in <module>()
----> 1 pd.read_csv('file_not_in_right_place.csv')

~/Envs/analysis/lib/python3.6/site-packages/pandas/io/parsers.py in parser_f(filepath_or_buffer, sep, delimiter, head
er, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_value
s, skipinitialspace, skiprows, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates,
infer_datetime_format, keep_date_col, date_parser, dayfirst, iterator, chunksize, compression, thousands, decimal, li
neterminator, quotechar, quoting, escapechar, comment, encoding, dialect, tupleize_cols, error_bad_lines, warn_bad_li
nes, skipfooter, doublequote, delim_whitespace, low_memory, memory_map, float_precision)
    676                     skip_blank_lines=skip_blank_lines)
    677
--> 678         return _read(filepath_or_buffer, kwds)
    679
    680     parser_f.__name__ = name

~/Envs/analysis/lib/python3.6/site-packages/pandas/io/parsers.py in _read(filepath_or_buffer, kwds)
    438
    439     # Create the parser.
--> 440     parser = TextFileReader(filepath_or_buffer, **kwds)
    441
    442     if chunksize or iterator:

~/Envs/analysis/lib/python3.6/site-packages/pandas/io/parsers.py in __init__(self, f, engine, **kwds)
    785             self.options['has_index_names'] = kwds['has_index_names']
    786
--> 787         self._make_engine(self.engine)
    788
    789     def close(self):
```

Pandas searches your 'current working directory' for the filename that you specify when opening or loading files. The FileNotFoundError can be due to a misspelled filename, or an incorrect working directory.

Finding your Python Path

Your Python path can be displayed using the built-in `os` module. The OS module is for operating system dependent functionality into Python programs and scripts.

To find your current working directory, the function required is `os.getcwd()`. The `os.listdir()` function can be used to display all files in a directory, which is a good check to see if the CSV file you are loading is in the directory as expected.

```
# Find out your current working directory
import os
print(os.getcwd())

# Out: /Users/shane/Documents/blog

# Display all of the files found in your current working directory
print(os.listdir(os.getcwd())


# Out: ['test_delimted.ssv', 'CSV Blog.ipynb', 'test_data.csv']
```

In the example above, my current working directory is in the '/Users/Shane/Document/blog' directory. Any files that are places in this directory wi

Confidentialité - Conditions

immediately available to the Python file open() function or the Pandas read csv function.

Instead of moving the required data files to your working directory, you can also change your current working directory to the directory where the files reside using `os.chdir()`.

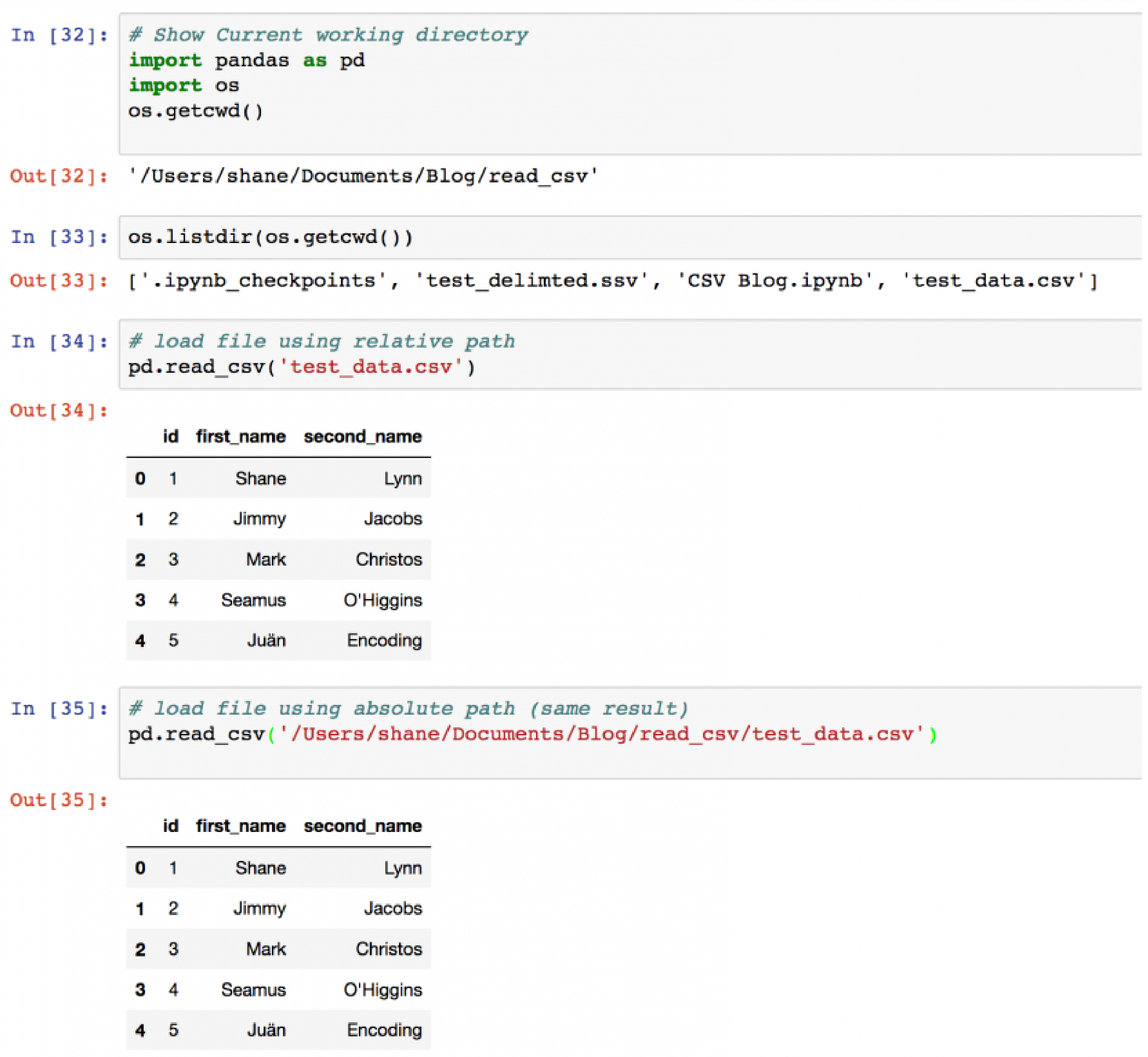
File Loading: Absolute and Relative Paths

When specifying file names to the read_csv function, you can supply both absolute or relative file paths.

- A **relative path** is the path to the file if you start from your current working directory. In relative paths, typically the file will be in a subdirectory of the working directory and the path will not start with a drive specifier, e.g. (*data/test_file.csv*). The characters '..' are used to move to a parent directory in a relative path.
- An **absolute path** is the complete path from the base of your file system to the file that you want to load, e.g. *c:/Documents/Shane/data/test_file.csv*. Absolute paths will start with a drive specifier (*c:/* or *d:/* in Windows, or '/' in Mac or Linux)

It's recommended and preferred to use relative paths where possible in applications, because absolute paths are unlikely to work on different computers due to different directory structures.

Confidentialité - Conditions



```
In [32]: # Show Current working directory
         import pandas as pd
         import os
         os.getcwd()

Out[32]: '/Users/shane/Documents/Blog/read_csv'

In [33]: os.listdir(os.getcwd())

Out[33]: ['.ipynb_checkpoints', 'test_delimted.ssv', 'CSV Blog.ipynb', 'test_data.csv']

In [34]: # load file using relative path
         pd.read_csv('test_data.csv')

Out[34]:
```

	id	first_name	second_name
0	1	Shane	Lynn
1	2	Jimmy	Jacobs
2	3	Mark	Christos
3	4	Seamus	O'Higgins
4	5	Juän	Encoding

```
In [35]: # load file using absolute path (same result)
         pd.read_csv('/Users/shane/Documents/Blog/read_csv/test_data.csv')

Out[35]:
```

	id	first_name	second_name
0	1	Shane	Lynn
1	2	Jimmy	Jacobs
2	3	Mark	Christos
3	4	Seamus	O'Higgins
4	5	Juän	Encoding

Loading the same file with Pandas read_csv using relative and absolute paths. Relative paths are directions to the file starting at your current working directory, where absolute paths always start at the base of your file system.

4. Pandas CSV File Loading Errors

The most common error's you'll get while loading data from CSV files into Pandas will be:

1. `FileNotFoundError: File b'filename.csv' does not exist`

   A File Not Found error is typically an issue with path setup, current directory, or file name confusion (file extension can play a part here!)

2. `UnicodeDecodeError: 'utf-8' codec can't decode byte in position : invalid continuation byte`

   A Unicode Decode Error is typically caused by not specifying the encoding of th

Confidentialité - Conditions

and happens when you have a file with non-standard characters. For a quick fix, try opening the file in Sublime Text, and re-saving with encoding 'UTF-8'.

3. `pandas.parser.CParserError: Error tokenizing data.`
   Parse Errors can be caused in unusual circumstances to do with your data format – try to add the parameter "engine='python'" to the read_csv function call; this changes the data reading function internally to a slower but more stable method.

Advanced CSV Loading

There are some additional flexible parameters in the Pandas read_csv() function that are useful to have in your arsenal of data science techniques:

Specifying Data Types

As mentioned before, CSV files do not contain any type information for data. Data types are inferred through examination of the top rows of the file, which can lead to errors. To manually specify the data types for different columns, the **dtype** parameter can be used with a dictionary of column names and data types to be applied, for example: `dtype={"name": str, "age": np.int32}`.

Note that for dates and date times, the format, columns, and other behaviour can be adjusted using **parse_dates**, **date_parser**, **dayfirst**, **keep_date** parameters.

Skipping and Picking Rows and Columns From File

The **nrows** parameter specifies how many rows from the top of CSV file to read, which is useful to take a sample of a large file without loading completely. Similarly the **skiprows** parameter allows you to specify rows to leave out, either at the start of the file (provide an int), or throughout the file (provide a list of row indices). Similarly, the **usecols** parameter can be used to specify which columns in the data to load.

Custom Missing Value Symbols

When data is exported to CSV from different systems, missing values can be specified with different tokens. The **na_values** parameter allows you to customise the characters that are recognised as missing values. The default values interpreted as NA/NaN are: '', '#N/A'

Confidentialité - Conditions

'#N/A N/A', '#NA', '-1.#IND', '-1.#QNAN', '-NaN', '-nan', '1.#IND', '1.#QNAN', 'N/A', 'NA', 'NULL', 'NaN', 'n/a', 'nan', 'null'.

```
# Advanced CSV loading example

data = pd.read_csv(
    "data/files/complex_data_example.tsv",      # relative python path to subdirectory
    sep='\t'           # Tab-separated value file.
    quotechar="'",        # single quote allowed as quote character
    dtype={"salary": int},             # Parse the salary column as an integer
    usecols=['name', 'birth_date', 'salary'].   # Only load the three columns specified.
    parse_dates=['birth_date'],     # Intepret the birth_date column as a date
    skiprows=10,         # Skip the first 10 rows of the file
    na_values=['.', '??']       # Take any '.' or '??' values as NA
)
```

CSV Format Positives and Negatives

As with all technical decisions, storing your data in CSV format has both advantages and disadvantages. Be aware of the potential pitfalls and issues that you will encounter as you load, store, and exchange data in CSV format:

On the plus side:

- CSV format is universal and the data can be loaded by almost any software.
- CSV files are simple to understand and debug with a basic text editor
- CSV files are quick to create and load into memory before analysis.

However, the CSV format has some negative sides:

- There is no data type information stored in the text file, all typing (dates, int vs float, strings) are inferred from the data only.
- There's no formatting or layout information storable – things like fonts, borders, column width settings from Microsoft Excel will be lost.
- File encodings can become a problem if there are non-ASCII compatible characters in text fields.

- CSV format is inefficient; numbers are stored as characters rather than binary values, which is wasteful. You will find however that your CSV data compresses well using zip compression.

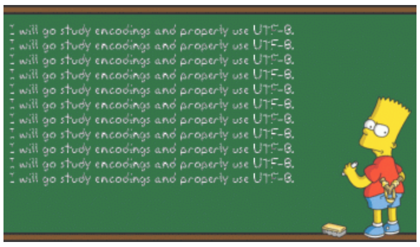
As and aside, in an effort to counter some of these disadvantages, two prominent data science developers in both the R and Python ecosystems, Wes McKinney and Hadley Wickham, recently introduced the Feather Format, which aims to be a fast, simple, open, flexible and multi-platform data format that supports multiple data types natively.

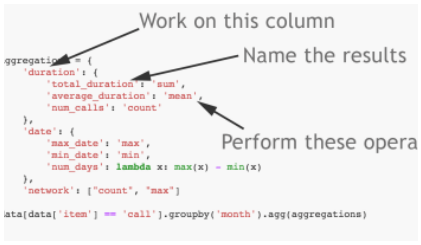
Additional Reading

1. **Official Pandas documentation** for the read_csv function.
2. **Python 3 Notes** on file paths, working directories, and using the OS module.
3. **Datacamp Tutorial on loading CSV files**, including some additional OS commands.
4. **PythonHow Loading CSV** tutorial.
5. **Chris Albon Notes on CSV loading** in Pandas.

Related

The Pandas DataFrame - loading, editing, and viewing data in Python
December 13, 2017
In "blog"


I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
I will go study encodings and properly use UTF-8.
Pandas CSV error: Error tokenizing data. C error: EOF inside string starting at line
May 21, 2017
In "blog"



```
ggregations = {
    'duration': {
        'total_duration': 'sum',
        'average_duration': 'mean',
        'num_calls': 'count'
    },
    'date': {
        'max_date': 'max',
        'min_date': 'min',
        'num_days': lambda x: max(x) - min(x)
    },
    'network': ["count", "max"]

ata[data['item'] == 'call'].groupby('month').agg(aggregations)
```

Summarising, Aggregating, and Grouping data in Python Pandas
June 14, 2015
In "blog"

← Previous Post Next Post →

6 thoughts on "Python Pandas read_csv – Load Data from CSV Fil