# CONTENTS

1 Introduction ......................................................... 3

2 Installation ....................................................... 5

3 Usage .................................................................
   3.1 Python Library .................................................. 7
   3.2 Command-Line Tool ............................................ 8
   3.3 Version Control Integration ................................. 10

4 Contributing ...................................................... 11

5 Notes .................................................................
   5.1 Quick Start ..................................................... 13
   5.2 Introduction ................................................... 14
   5.3 Installation .................................................... 15
   5.4 Usage .......................................................... 15
   5.5 Contributing ................................................... 19
   5.6 Notes ............................................................ 19
   5.7 Changelog ....................................................... 19
   5.8 CleverCSV API Documentation ............................. 25

Python Module Index ................................................ 41

Index ........................................................................ 43
CleverCSV provides a drop-in replacement for the Python csv package with improved dialect detection for messy CSV files. It also provides a handy command line tool that can standardize a messy file or generate Python code to import it.

Useful links:
- CleverCSV on Github
- CleverCSV on PyPI
- Demo of CleverCSV on Binder (interactive!)
- Research Paper on CSV dialect detection (PDF)
- Reproducible Research Repo
- Blog post on messy CSV files
- Discussion forum: a place to ask questions and share ideas!

Click here to go to the introduction with more details about CleverCSV. If you’re in a hurry, below is a quick overview of how to get started with the CleverCSV Python package and the command line interface.

For the Python package:

```python
# Import the package
>>> import clevercsv

# Load the file as a list of rows
# This uses the imdb.csv file in the examples directory
>>> rows = clevercsv.read_table('./imdb.csv')

# Load the file as a Pandas Dataframe
# Note that df = pd.read_csv('./imdb.csv') would fail here
>>> df = clevercsv.read_dataframe('./imdb.csv')

# Use CleverCSV as drop-in replacement for the Python CSV module
# This follows the Sniffer example: https://docs.python.org/3/library/csv.html#csv.Sniffer
>>> with open('./imdb.csv', newline='') as csvfile:
...     dialect = clevercsv.Sniffer().sniff(csvfile.read())
...     csvfile.seek(0)
...     reader = clevercsv.reader(csvfile, dialect)
...     rows = list(reader)
```

And for the command line interface:

```bash
# Install the full version of CleverCSV (this includes the command line interface)
$ pip install clevercsv[full]

# Detect the dialect
$ clevercsv detect ./imdb.csv
Detected: SimpleDialect( ',', '', '
')
```

(continues on next page)
# Generate code to import the file
$ clevercsv code ./imdb.csv

import clevercsv

with open("./imdb.csv", "r", newline="", encoding="utf-8") as fp:
    reader = clevercsv.reader(fp, delimiter=",", quotechar="", escapechar="\")
    rows = list(reader)

# Explore the CSV file as a Pandas dataframe
$ clevercsv explore -p imdb.csv
Dropping you into an interactive shell.
CleverCSV has loaded the data into the variable: df

>>> df
• CSV files are awesome! They are lightweight, easy to share, human-readable, version-controllable, and supported by many systems and tools!

• CSV files are terrible! They can have many different formats, multiple tables, headers or no headers, escape characters, and there’s no support for recording metadata!

CleverCSV is a Python package that aims to solve some of the pain points of CSV files, while maintaining many of the good things. The package automatically detects (with high accuracy) the format (*dialect*) of CSV files, thus making it easier to simply point to a CSV file and load it, without the need for human inspection. In the future, we hope to solve some of the other issues of CSV files too.

CleverCSV is based on science. We investigated thousands of real-world CSV files to find a robust way to automatically detect the dialect of a file. This may seem like an easy problem, but to a computer a CSV file is simply a long string, and every dialect will give you *some* table. In CleverCSV we use a technique based on the patterns of row lengths of the parsed file and the data type of the resulting cells. With our method we achieve 97% accuracy for dialect detection, with a 21% improvement on non-standard (*messy*) CSV files compared to the Python standard library.

We think this kind of work can be very valuable for working data scientists and programmers and we hope that you find CleverCSV useful (if there’s a problem, please open an issue!) Since the academic world counts citations, please cite CleverCSV if you use the package. Here’s a BibTeX entry you can use:

```bibtex
@article{van2019wrangling,
  title = {Wrangling Messy {CSV} Files by Detecting Row and Type Patterns},
  author = {{van den Burg}, G. J. J. and Naz'abal, A. and Sutton, C.},
  journal = {Data Mining and Knowledge Discovery},
  year = {2019},
  volume = {33},
  number = {6},
  pages = {1799--1820},
  issn = {1573-756X},
  doi = {10.1007/s10618-019-00646-y},
}
```

And of course, if you like the package please spread the word! You can do this by Tweeting about it (#CleverCSV) or clicking the on GitHub!
CleverCSV is available on PyPI. You can install either the full version, which includes the command line interface and all optional dependencies, using

```bash
$ pip install clevercsv[full]
```

or you can install a lighter, core version of CleverCSV with

```bash
$ pip install clevercsv
```
CleverCSV consists of a Python library and a command line tool called clevercsv.

### 3.1 Python Library

We designed CleverCSV to provide a drop-in replacement for the built-in CSV module, with some useful functionality added to it. Therefore, if you simply want to replace the built-in CSV module with CleverCSV, you can import CleverCSV as follows, and use it as you would use the built-in csv module.

```python
import clevercsv
```

CleverCSV provides an improved version of the dialect sniffer in the CSV module, but it also adds some useful wrapper functions. These functions automatically detect the dialect and aim to make working with CSV files easier. We currently have the following helper functions:

- `detect_dialect`: takes a path to a CSV file and returns the detected dialect
- `read_table`: automatically detects the dialect and encoding of the file, and returns the data as a list of rows. A version that returns a generator is also available: `stream_table`
- `read_dataframe`: detects the dialect and encoding of the file and then uses Pandas to read the CSV into a DataFrame. Note that this function requires Pandas to be installed.
- `read_dicts`: detect the dialect and return the rows of the file as dictionaries, assuming the first row contains the headers. A streaming version called `stream_dicts` is also available.
- `write_table`: write a table (a list of lists) to a file using the RFC-4180 dialect.

Of course, you can also use the traditional way of loading a CSV file, as in the Python CSV module:

```python
import clevercsv

with open("data.csv", "r", newline="") as fp:
    dialect = clevercsv.Sniffer().sniff(fp.read(), verbose=False)
    fp.seek(0)
    reader = clevercsv.reader(fp, dialect)
    rows = list(reader)
```

For large files, you can speed up detection by supplying a smaller sample to the sniffer, for instance:

```python
dialect = clevercsv.Sniffer().sniff(fp.read(10000))
```
That’s the basics! If you want more details, you can look at the code of the package, the test suite, or the API documentation. If you run into any issues or have comments or suggestions, please open an issue on GitHub.

3.2 Command-Line Tool

To use the command line tool, make sure that you install the full version of CleverCSV (see above).

The clevercsv command line application has a number of handy features to make working with CSV files easier. For instance, it can be used to view a CSV file on the command line while automatically detecting the dialect. It can also generate Python code for importing data from a file with the correct dialect. The full help text is as follows:

```
USAGE
    clevercsv [-h] [-v] [-V] <command> [arg1] ... [argN]

ARGUMENTS
    <command> The command to execute
    <arg> The arguments of the command

GLOBAL OPTIONS
    -h (--help) Display this help message.
    -v (--verbose) Enable verbose mode.
    -V (--version) Display the application version.

AVAILABLE COMMANDS
    code Generate Python code for importing the CSV file
    detect Detect the dialect of a CSV file
    explore Drop into a Python shell with the CSV file loaded
    help Display the manual of a command
    standardize Convert a CSV file to one that conforms to RFC-4180
    view View the CSV file on the command line using TabView
```

Each of the commands has further options (for instance, the code and explore commands have support for importing the CSV file as a Pandas DataFrame). Use clevercsv help <command> for more information. Below are some examples for each command.

Note that each command accepts the -n or --num-chars flag to set the number of characters used to detect the dialect. This can be especially helpful to speed up dialect detection on large files.

3.2.1 Code

Code generation is useful when you don’t want to detect the dialect of the same file over and over again. You simply run the following command and copy the generated code to a Python script!

```
$ clevercsv code imdb.csv

# Code generated with CleverCSV

import clevercsv

with open("imdb.csv", "r", newline='', encoding="utf-8") as fp:
    reader = clevercsv.reader(fp, delimiter=",", quotechar="'", escapechar="\\")
    rows = list(reader)
```
We also have a version that reads a Pandas dataframe:

```bash
$ clevercsv code --pandas imdb.csv
# Code generated with CleverCSV
import clevercsv
df = clevercsv.read_dataframe("imdb.csv", delimiter="", quotechar='', escapechar='\\')
```

### 3.2.2 Detect

Detection is useful when you only want to know the dialect.

```bash
$ clevercsv detect imdb.csv
Detected: SimpleDialect('',' ','')
```

The `--plain` flag gives the components of the dialect on separate lines, which makes combining it with `grep` easier.

```bash
$ clevercsv detect --plain imdb.csv
delimiter = ,
quotechar = 
escapechar = \
```

### 3.2.3 Explore

The `explore` command is great for a command-line based workflow, or when you quickly want to start working with a CSV file in Python. This command detects the dialect of a CSV file and starts an interactive Python shell with the file already loaded! You can either have the file loaded as a list of lists:

```bash
$ clevercsv explore milk.csv
Dropping you into an interactive shell.

CleverCSV has loaded the data into the variable: rows
```

```python
>>> len(rows)
381
```

or you can load the file as a Pandas dataframe:

```bash
$ clevercsv explore -p imdb.csv
Dropping you into an interactive shell.

CleverCSV has loaded the data into the variable: df
```

```python
>>> df.head()
```

<table>
<thead>
<tr>
<th>fn</th>
<th>tid</th>
<th>...</th>
<th>War</th>
<th>Western</th>
</tr>
</thead>
<tbody>
<tr>
<td>titles01/tt0012349</td>
<td>tt0012349</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>titles01/tt0015864</td>
<td>tt0015864</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>titles01/tt0017136</td>
<td>tt0017136</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>titles01/tt0017925</td>
<td>tt0017925</td>
<td></td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(continues on next page)
3.2.4 Standardize

Use the standardize command when you want to rewrite a file using the RFC-4180 standard:

```bash
$ clevercsv standardize --output imdb_standard.csv imdb.csv
```

In this particular example the use of the escape character is replaced by using quotes.

3.2.5 View

This command allows you to view the file in the terminal. The dialect is of course detected using CleverCSV! Both this command and the standardize command support the --transpose flag, if you want to transpose the file before viewing or saving:

```bash
$ clevercsv view --transpose imdb.csv
```

3.3 Version Control Integration

If you’d like to make sure that you never commit a messy (non-standard) CSV file to your repository, you can install a pre-commit hook. First, install pre-commit using the installation instructions. Next, add the following configuration to the .pre-commit-config.yaml file in your repository:

```
repos:
  - repo: https://github.com/alan-turing-institute/CleverCSV-pre-commit
    rev: v0.6.6  # or any later version
    hooks:
      - id: clevercsv-standardize
```

Finally, run pre-commit install to set up the git hook. Pre-commit will now use CleverCSV to standardize your CSV files following RFC-4180 whenever you commit a CSV file to your repository.
If you want to encourage development of CleverCSV, the best thing to do now is to *spread the word!*

If you encounter an issue in CleverCSV, please open an issue or submit a pull request. Don’t hesitate, you’re helping to make this project better for everyone! If GitHub’s not your thing but you still want to contact us, you can send an email to gertjanvandenburg at gmail dot com instead. You can also ask questions on Gitter.

Note that all contributions to the project must adhere to the Code of Conduct.

The CleverCSV package was originally written by Gertjan van den Burg and came out of scientific research on wrangling messy CSV files by Gertjan van den Burg, Alfredo Nazabal, and Charles Sutton.
CleverCSV is licensed under the MIT license. Please cite our research if you use CleverCSV in your work.

Copyright (c) 2018-2021 The Alan Turing Institute.

CleverCSV provides a drop-in replacement for the Python csv package with improved dialect detection for messy CSV files. It also provides a handy command line tool that can standardize a messy file or generate Python code to import it.

Useful links:
- CleverCSV on Github
- CleverCSV on PyPI
- Demo of CleverCSV on Binder (interactive!)
- Research Paper on CSV dialect detection (PDF)
- Reproducible Research Repo
- Blog post on messy CSV files
- Discussion forum: a place to ask questions and share ideas!

Contents: | | | | | | | |

5.1 Quick Start

Click here to go to the introduction with more details about CleverCSV. If you’re in a hurry, below is a quick overview of how to get started with the CleverCSV Python package and the command line interface.

For the Python package:

```
# Import the package
>>> import clevercsv

# Load the file as a list of rows
# This uses the imdb.csv file in the examples directory
>>> rows = clevercsv.read_table('./imdb.csv')

# Load the file as a Pandas Dataframe
```

(continues on next page)
5.2 Introduction

- CSV files are awesome! They are lightweight, easy to share, human-readable, version-controllable, and supported by many systems and tools!

- CSV files are terrible! They can have many different formats, multiple tables, headers or no headers, escape characters, and there’s no support for recording metadata!

CleverCSV is a Python package that aims to solve some of the pain points of CSV files, while maintaining many of the good things. The package automatically detects (with high accuracy) the format (dialect) of CSV files, thus making it easier to simply point to a CSV file and load it, without the need for human inspection. In the future, we hope to solve some of the other issues of CSV files too.

CleverCSV is based on science. We investigated thousands of real-world CSV files to find a robust way to automatically detect the dialect of a file. This may seem like an easy problem, but to a computer a CSV file is simply a long string,
and every dialect will give you some table. In CleverCSV we use a technique based on the patterns of row lengths of the parsed file and the data type of the resulting cells. With our method we achieve 97% accuracy for dialect detection, with a 21% improvement on non-standard (messy) CSV files compared to the Python standard library.

We think this kind of work can be very valuable for working data scientists and programmers and we hope that you find CleverCSV useful (if there’s a problem, please open an issue!) Since the academic world counts citations, please cite CleverCSV if you use the package. Here’s a BibTeX entry you can use:

```bibtex
@article{van2019wrangling,
  title = {Wrangling Messy \{CSV\} Files by Detecting Row and Type Patterns},
  author = {{van den Burg}, G. J. J. and Naz'a bal, A. and Sutton, C.},
  journal = {Data Mining and Knowledge Discovery},
  year = {2019},
  volume = {33},
  number = {6},
  pages = {1799--1820},
  issn = {1573-756X},
  doi = {10.1007/s10618-019-00646-y},
}
```

And of course, if you like the package please spread the word! You can do this by Tweeting about it (#CleverCSV) or clicking the on GitHub!

## 5.3 Installation

CleverCSV is available on PyPI. You can install either the full version, which includes the command line interface and all optional dependencies, using

```
$ pip install clevercsv[full]
```

or you can install a lighter, core version of CleverCSV with

```
$ pip install clevercsv
```

## 5.4 Usage

CleverCSV consists of a Python library and a command line tool called clevercsv.

### 5.4.1 Python Library

We designed CleverCSV to provide a drop-in replacement for the built-in CSV module, with some useful functionality added to it. Therefore, if you simply want to replace the builtin CSV module with CleverCSV, you can import CleverCSV as follows, and use it as you would use the builtin csv module.

```python
import clevercsv
```

CleverCSV provides an improved version of the dialect sniffer in the CSV module, but it also adds some useful wrapper functions. These functions automatically detect the dialect and aim to make working with CSV files easier. We currently have the following helper functions:

- `detect_dialect`: takes a path to a CSV file and returns the detected dialect
• **read_table**: automatically detects the dialect and encoding of the file, and returns the data as a list of rows. A version that returns a generator is also available: `stream_table`  
• **read_dataframe**: detects the dialect and encoding of the file and then uses Pandas to read the CSV into a DataFrame. Note that this function requires Pandas to be installed.  
• **read_dicts**: detect the dialect and return the rows of the file as dictionaries, assuming the first row contains the headers. A streaming version called `stream_dicts` is also available.  
• **write_table**: write a table (a list of lists) to a file using the RFC-4180 dialect.

Of course, you can also use the traditional way of loading a CSV file, as in the Python CSV module:

```python
import clevercsv

with open("data.csv", "r", newline="") as fp:
    dialect = clevercsv.Sniffer().sniff(fp.read(), verbose=False)
    fp.seek(0)
    reader = clevercsv.reader(fp, dialect)
    rows = list(reader)
```

For large files, you can speed up detection by supplying a smaller sample to the sniffer, for instance:

```python
dialect = clevercsv.Sniffer().sniff(fp.read(10000))
```

That’s the basics! If you want more details, you can look at the code of the package, the test suite, or the API documentation. If you run into any issues or have comments or suggestions, please open an issue on GitHub.

### 5.4.2 Command-Line Tool

*To use the command line tool, make sure that you install the full version of CleverCSV (see above).*

The `clevercsv` command line application has a number of handy features to make working with CSV files easier. For instance, it can be used to view a CSV file on the command line while automatically detecting the dialect. It can also generate Python code for importing data from a file with the correct dialect. The full help text is as follows:

**Usage**

```
USAGE
clevercsv [-h] [-v] [-V] <command> [<arg1>] ... [<argN>]
```

**Arguments**

```
ARGUMENTS
<command> The command to execute
<arg> The arguments of the command
```

**Global Options**

```
GLOBAL OPTIONS
-h (--help) Display this help message.
-v (--verbose) Enable verbose mode.
-V (--version) Display the application version.
```

**Available Commands**

```
AVAILABLE COMMANDS
code Generate Python code for importing the CSV file
detect Detect the dialect of a CSV file
drop Drop into a Python shell with the CSV file loaded
display Display the manual of a command
standardize Convert a CSV file to one that conforms to RFC-4180
view View the CSV file on the command line using TabView
```
Each of the commands has further options (for instance, the code and explore commands have support for importing the CSV file as a Pandas DataFrame). Use clevercsv help <command> for more information. Below are some examples for each command.

Note that each command accepts the -n or --num-chars flag to set the number of characters used to detect the dialect. This can be especially helpful to speed up dialect detection on large files.

**Code**

Code generation is useful when you don't want to detect the dialect of the same file over and over again. You simply run the following command and copy the generated code to a Python script!

```bash
$ clevercsv code imdb.csv

# Code generated with CleverCSV

import clevercsv

with open("imdb.csv", "r", newline="", encoding="utf-8") as fp:
    reader = clevercsv.reader(fp, delimiter="", quotechar="", escapechar="\")
    rows = list(reader)
```

We also have a version that reads a Pandas dataframe:

```bash
$ clevercsv code --pandas imdb.csv

# Code generated with CleverCSV

import clevercsv

df = clevercsv.read_dataframe("imdb.csv", delimiter="", quotechar="", escapechar="\")
```

**Detect**

Detection is useful when you only want to know the dialect.

```bash
$ clevercsv detect imdb.csv
Detected: SimpleDialect(’,’, ‘’, ‘\’)
```

The --plain flag gives the components of the dialect on separate lines, which makes combining it with grep easier.

```bash
$ clevercsv detect --plain imdb.csv
delimiter = ,
quotechar =
escapechar = \
```
Explore

The `explore` command is great for a command-line based workflow, or when you quickly want to start working with a CSV file in Python. This command detects the dialect of a CSV file and starts an interactive Python shell with the file already loaded! You can either have the file loaded as a list of lists:

```
$ clevercsv explore milk.csv
Dropping you into an interactive shell.
CleverCSV has loaded the data into the variable: rows
>>> len(rows)
381
```

or you can load the file as a Pandas dataframe:

```
$ clevercsv explore -p imdb.csv
Dropping you into an interactive shell.
CleverCSV has loaded the data into the variable: df
>>> df.head()
fn tid ... War Western
0 titles01/tt0012349 tt0012349 ... 0 0
1 titles01/tt0015864 tt0015864 ... 0 0
2 titles01/tt0017136 tt0017136 ... 0 0
3 titles01/tt0017925 tt0017925 ... 0 0
4 titles01/tt0021749 tt0021749 ... 0 0
```

Standardize

Use the `standardize` command when you want to rewrite a file using the RFC-4180 standard:

```
$ clevercsv standardize --output imdb_standard.csv imdb.csv
```

In this particular example the use of the escape character is replaced by using quotes.

View

This command allows you to view the file in the terminal. The dialect is of course detected using CleverCSV! Both this command and the `standardize` command support the `--transpose` flag, if you want to transpose the file before viewing or saving:

```
$ clevercsv view --transpose imdb.csv
```
5.4.3 Version Control Integration

If you’d like to make sure that you never commit a messy (non-standard) CSV file to your repository, you can install a pre-commit hook. First, install pre-commit using the installation instructions. Next, add the following configuration to the .pre-commit-config.yaml file in your repository:

```
repos:
  - repo: https://github.com/alan-turing-institute/CleverCSV-pre-commit
    rev: v0.6.6 # or any later version
    hooks:
      - id: clevercsv-standardize
```

Finally, run pre-commit install to set up the git hook. Pre-commit will now use CleverCSV to standardize your CSV files following RFC-4180 whenever you commit a CSV file to your repository.

5.5 Contributing

If you want to encourage development of CleverCSV, the best thing to do now is to spread the word!

If you encounter an issue in CleverCSV, please open an issue or submit a pull request. Don’t hesitate, you’re helping to make this project better for everyone! If GitHub’s not your thing but you still want to contact us, you can send an email to gertjanvandenburg at gmail dot com instead. You can also ask questions on Gitter.

Note that all contributions to the project must adhere to the Code of Conduct.

The CleverCSV package was originally written by Gertjan van den Burg and came out of scientific research on wrangling messy CSV files by Gertjan van den Burg, Alfredo Nazabal, and Charles Sutton.

5.6 Notes

CleverCSV is licensed under the MIT license. Please cite our research if you use CleverCSV in your work.

Copyright (c) 2018-2021 The Alan Turing Institute.

5.7 Changelog

5.7.1 Version 0.7.0

- Add a JSON object data type to address a specific failure case (#37).
- Add support for timezones for time data type
- Add support for building wheels on non-native architectures (#39).
- Add a flag to disable skipping type detection using the command line interface.
5.7.2 Version 0.6.8

• Add a “bytearray” type to address a specific failure case (#35).
• Minor clarifications to licensing.

5.7.3 Version 0.6.7

• Updates to release process. This version introduces pre-compiled wheels for Python 3.9.

5.7.4 Version 0.6.6

• Add an encoding argument to write_table to allow specifying the output encoding. Thanks to @mitchgrogg for reporting issue #27.

5.7.5 Version 0.6.5

• Add support for standardizing in-place and standardizing multiple files.
• Add warning on duplicate field names in DictReader
• Add return value to writers to match the standard library.

5.7.6 Version 0.6.4

• Various speed ups to constructing the list of potential dialects. This removes a costly step of the detection process that will likely add a few more potential dialects, but has the end result of making overall dialect detection faster.

5.7.7 Version 0.6.3

• Rename wrapper functions to a more coherent naming scheme. Old names will be available until 0.7.0, but now produce a FutureWarning.
• Add stream_dicts wrapper function.
• Improve handling of file encoding for the read_dataframe wrapper: detected encoding is now passed on to Pandas.
• Fix handling of optional dependency error for TabView on non-Windows platforms.

5.7.8 Version 0.6.2

• Update URL regex to avoid catastrophic backtracking and increase performance. See issue #13 and issue #15. Thanks to @kaskawu for the fix and @jlumbroso for re-raising the issue.
• Add num_chars keyword argument to read_as_dicts and csv2df wrappers.
• Improve documentation w.r.t. handling large files. Thanks to @jlumbroso for raising this issue.
5.7.9 Version 0.6.1

• Add an `explore` command to the command line application for CleverCSV. This command makes it easy to start exploring a CSV file using the Python interactive shell.

5.7.10 Version 0.6.0

• Split the package into a “core” and “full” version. This allows users who only need the improved dialect detection functionality to download a version with a smaller footprint. Fixes issue #10. Thanks to @seperman.

5.7.11 Version 0.5.6

• Fix speed of `unix_path` regex used in type detection. (issue #13). Thanks to @kaskawu.

5.7.12 Version 0.5.5

• Add `stream_csv` wrapper that returns a generator over rows
• Minor update to the URL type detection
• Documentation updates

5.7.13 Version 0.5.4

• Fix bugs discovered from fuzz testing (issue #7)
• Minor changes to readme and code quality

5.7.14 Version 0.5.3

• Fix using `nan` as default value when skipping a dialect (issue #5)

5.7.15 Version 0.5.2

• Bump version to fix wheel building

5.7.16 Version 0.5.1

• Bump version to fix wheel building
5.7.17 Version 0.5.0

- Improve type detection for quoted alphanumeric cells (#4)
- Pass strict dialect property to parser.

5.7.18 Version 0.4.7

- Bugfix for write_table wrapper on Windows.
- Move building Windows platform wheels to Travis.
- Use cibuildwheel version 1.0.0 for building wheels.

5.7.19 Version 0.4.6

- Add a wrapper function that writes a table to a CSV file.

5.7.20 Version 0.4.5

- Update CleverCSV to match updated clikit dependency
- Fix dependency versions for clikit and cleo

5.7.21 Version 0.4.4

- Update standardize command to use CRLF line endings on all platforms.
- Add workaround for Tabview being unavailable on Windows.
- Remove packaging and dependency management with Poetry.
- Add support for building platform wheels on Travis and AppVeyor.

5.7.22 Version 0.4.3

- Add optional method parameter to dialect detector.
- Bugfix for clevercsv code command when the delimiter is tab.

5.7.23 Version 0.4.2

- Fix a failing build due to dependency version mismatch
5.7.24 Version 0.4.1

- Allow underscore in alphanumeric strings
- Update unix path regular expression
- Add more integration tests and log detection method

5.7.25 Version 0.4.0

- Update URL regular expression and add unit tests
- Add IPv4 type detection
- Add tie-breaker for combined quotechar and escapechar ties

5.7.26 Version 0.3.7

- Bugfix for console script code command
- Update readme

5.7.27 Version 0.3.6

- Cleanly handle failure to detect dialect in console application
- Remove any (partial) support for Python 2

5.7.28 Version 0.3.5

- Remove Python parser - this speeds up file reading and tie breaking

5.7.29 Version 0.3.4

- Ensure the C parser is used in the reader.
- Update integration tests to improve error handling
- Readme updates

5.7.30 Version 0.3.3

- Ensure detected encoding is in the generated Python code for the clevercsv code command.
- Ensure encoding is detected in wrappers.detect_dialect.
- Bugfix in integration test
- Expand readme
5.7.31 Version 0.3.2

- Add documentation on Read the Docs
- Use requirements.txt file for dependencies when packaging

5.7.32 Version 0.3.1

- Add help description to each CLI command
- Update README
- Add transpose flag for standardize and view commands

5.7.33 Version 0.3.0

- Rewrite console application using Cleo
- Add unit tests for console application
- Add detect_dialect wrapper function
- Add support for “unix_path” data type in type detection
- Add encoding and num_chars options to read_csv wrapper
- Add -p/--pandas flag to code command to generate Pandas output.

5.7.34 Version 0.2.5

- Rename read_as_lol to read_csv.

5.7.35 Version 0.2.4

- Allow setting the number of characters to read
- Simplify printing of skipped potential dialects

5.7.36 Version 0.2.3

- Add read_as_lol wrapper function.

5.7.37 Version 0.2.2

- Add code command to clevercsv command line program.
5.7.38 Version 0.2.1

• Bugfix to update executable to new name

5.7.39 Version 0.2.0

• Rename package to clevercsv

5.8 CleverCSV API Documentation

5.8.1 clevercsv package

Subpackages

clevercsv.console package

Subpackages

clevercsv.console.commands package

Submodules

clevercsv.console.commands.code module
clevercsv.console.commands.detect module
clevercsv.console.commands.explore module
clevercsv.console.commands.standardize module
clevercsv.console.commands.view module

Module contents

Submodules

clevercsv.console.application module
clevercsv.console.config module

Module contents

Submodules
clevercsv.break_ties module

Break ties in the data consistency measure.
Author: Gertjan van den Burg

clevercsv.break_ties.break_ties_four(data, dialects)

Break ties between four dialects.

This function works by breaking the ties between pairs of dialects that result in the same parsing result (if any).
If this reduces the number of dialects, then break_ties_three() or break_ties_two() is used, otherwise, the tie can't be broken.

Ties are only broken if all dialects have the same delimiter.

Parameters
• data (str) – The data of the file as a string
• dialects (list) – List of SimpleDialect objects

Returns dialect – The chosen dialect if the tie can be broken, None otherwise.

Return type SimpleDialect

Notes
We have only observed one case during development where this function was needed. It may need to be revisited in the future if other examples are found.

clevercsv.break_ties.break_ties_three(data, A, B, C)

Break ties between three dialects.

If the delimiters and the escape characters are all equal, then we look for the dialect that has no quotechar. The tie is broken by calling break_ties_two() for the dialect without quotechar and another dialect that gives the same parsing result.
If only the delimiter is the same for all dialects then use break_ties_two() on the dialects that do not have a quotechar, provided there are only two of these.

Parameters
• data (str) – The data of the file as a string
• A (SimpleDialect) – a dialect
• B (SimpleDialect) – a dialect
• C (SimpleDialect) – a dialect

Returns dialect – The chosen dialect if the tie can be broken, None otherwise.

Return type SimpleDialect
Notes

We have only observed one tie for each case during development, so this may need to be improved in the future.

clevercsv.break_ties.break_ties_two(data, A, B)

Break ties between two dialects.

This function breaks ties between two dialects that give the same score. We distinguish several cases:

1. If delimiter and escapechar are the same and one of the quote characters is the empty string. We parse the file with both dialects and check if the parsing result is the same. If it is, the correct dialect is the one with no quotechar, otherwise it’s the other one. 2. If quotechar and escapechar are the same and the delimiters are comma and space, then we go for comma. Alternatively, if either of the delimiters is the hyphen, we assume it’s the other dialect. 3. If the delimiter and quotechar is the same and one dialect uses the escapchar and the other doesn’t. We break this tie by checking if the escapechar has an effect and if it occurs an even or odd number of times.

If it’s none of these cases, we don’t break the tie and return None.

Parameters

- **data** (str) – The data of the file as a string.
- **A** (SimpleDialect) – A potential dialect
- **B** (SimpleDialect) – A potential dialect

Returns **dialect** – The chosen dialect if the tie can be broken, None otherwise.

Return type **SimpleDialect** or None

clevercsv.break_ties.reduce_pairwise(data, dialects)

Reduce the set of dialects by breaking pairwise ties

Parameters

- **data** (str) – The data of the file as a string
- **dialects** (list) – List of SimpleDialect objects

Returns **dialects** – List of SimpleDialect objects.

Return type **list**

clevercsv.break_ties.tie_breaker(data, dialects)

Break ties between dialects.

This function is used to break ties where possible between two, three, or four dialects that receive the same value for the data consistency measure.

Parameters

- **data** (str) – The data as a single string
- **dialects** (list) – Dialects that are tied

Returns **dialect** – One of the dialects from the list provided or None.

Return type **SimpleDialect**
**clevercsv.consistency module**

Detect the dialect using the data consistency measure.

Author: Gertjan van den Burg

```python
clevercsv.consistency.break_ties(data, dialects)
clevercsv.consistency.consistency_scores(data, dialects, skip=True, logger=<built-in function print>)
clevercsv.consistency.detect_consistency_dialects(data, dialects, skip=True, verbose=False)
```

Wrapper for dialect detection with the consistency measure

This function takes a list of dialects to consider.

```python
clevercsv.consistency.detect_dialect_consistency(data, delimiters=None, skip=True, verbose=False)
```

Detect the dialect with the data consistency measure

This uses the data consistency measure to detect the dialect. See the paper for details.

**Parameters**

- `data (str)` – The data of the file as a string
- `delimiters (iterable)` – List of delimiters to consider. If None, the `get_delimiters()` function is used to automatically detect this (as described in the paper).
- `skip (bool)` – Skip computation of the type score for dialects with a low pattern score.
- `verbose (bool)` – Print out the dialects considered and their scores.

**Returns**

- `dialect` – The detected dialect. If no dialect could be detected, returns None.

**Return type** `SimpleDialect`

```python
clevercsv.consistency.get_best_set(scores)
```

**clevercsv.cparser_util module**

Python utility functions that wrap the C parser.

```python
clevercsv.cparser_util.field_size_limit(*args, **kwargs)
```

Get/Set the limit to the field size.

This function is adapted from the one in the Python CSV module. See the documentation there.

```python
clevercsv.cparser_util.parse_data(data, dialect=None, delimiter=None, quotechar=None, escapechar=None, strict=None, return_quoted=False)
```

Parse the data given a dialect using the C parser.

**Parameters**

- `data (iterable)` – The data of the CSV file as an iterable
- `dialect (SimpleDialect)` – The dialect to use for the parsing. If None, the dialect with each component set to the empty string is used.
- `delimiter (str)` – The delimiter to use. If not None, overwrites the delimiter in the dialect.
- `quotechar (str)` – The quote character to use. If not None, overwrites the quote character in the dialect.
- `escapechar (str)` – The escape character to use. If not None, overwrites the escape character in the dialect.
• **strict** *(bool)* – Enable strict mode or not. If not None, overwrites the strict mode set in the dialect.

• **return_quoted** *(bool)* – For each cell, return a tuple “(field, is_quoted)” where the second element indicates whether the cell was a quoted cell or not.

**Yields rows** *(list)* – The rows of the file as a list of cells.

**Raises** *[Error]* – When an error occurs during parsing.

clevercsv.cparser_util.parse_string(*data*, *args*, **kwargs*)

Utility for when the CSV file is encoded as a single string

clevercsv.detect module

Drop-in replacement for Python Sniffer object.

Author: Gertjan van den Burg

**class** clevercsv.detect.Detector

**Bases:** object

Detect the Dialect of CSV files with normal forms or the data consistency measure. This class provides a drop-in replacement for the Python dialect Sniffer from the standard library.

**Note:** We call the object Detector just to mark the difference in the implementation and avoid naming issues. You can import it as **from ccsv import Sniffer** nonetheless.

detect(*sample*, *delimiters=None*, *verbose=False*, *method='auto'*, *skip=True*)

**has_header** *(sample)*

Detect if a file has a header from a sample.

This function is copied from CPython! The only change we’ve made is to use our dialect detection method.

sni ff(*sample*, *delimiters=None*, *verbose=False*)

clevercsv.detect_pattern module

Code for computing the pattern score.

Author: Gertjan van den Burg

clevercsv.detect_pattern.fill_empties(*abstract*)

Fill empty cells in the abstraction

The way the row patterns are constructed assumes that empty cells are marked by the letter C as well. This function fill those in. The function also removes duplicate occurrences of CC and replaces these with C.

**Parameters** abstract *(str)* – The abstract representation of the file.

**Returns** abstraction – The abstract representation with empties filled.

**Return type** str

clevercsv.detect_pattern.make_abstraction(*data*, *dialect*)

Create an abstract representation of the CSV file based on the dialect.

This function constructs the basic abstraction used to compute the row patterns.

**Parameters**
• **data** *(str)* – The data of the file as a string.

• **dialect** *(SimpleDialect)* – A dialect to parse the file with.

**Returns** abstraction – An abstract representation of the CSV file.

**Return type** *str*

clevercsv.detect_pattern.merge_with_quotechar(*S, dialect*)

Merge quoted blocks in the abstraction

This function takes the abstract representation and merges quoted blocks *(QC...CQ)* to a single cell *(C)*. The function takes nested quotes into account.

**Parameters**

• **S** *(str)* – The data of a file as a string

• **dialect** *(SimpleDialect)* – The dialect used to make the abstraction.

**Returns** abstraction – A simplified version of the abstraction with quoted blocks merged.

**Return type** *str*

clevercsv.detect_pattern.pattern_score(*data, dialect, eps=0.001*)

Compute the pattern score for given data and a dialect.

**Parameters**

• **data** *(string)* – The data of the file as a raw character string

• **dialect** *(dialect.Dialect)* – The dialect object

**Returns** score – the pattern score

**Return type** *float*

clevercsv.detect_pattern.strip_trailing(*abstract*)

Strip trailing row separator from abstraction.

**clevercsv.detect_type module**

Code for computing the type score.

Author: Gertjan van den Burg

**class** clevercsv.detect_type.TypeDetector(*strip_whitespace=True*)

**Bases:** object

**detect_type** *(cell, is_quoted=False)*

**is_bytearray** *(cell: str, **kwargs) → bool*

**is_currency** *(cell, **kwargs)*

**is_date** *(cell, **kwargs)*

**is_datetime** *(cell, **kwargs)*

**is_email** *(cell, **kwargs)*

**is_empty** *(cell, **kwargs)*

**is_ipv4** *(cell, **kwargs)*

**is_json_obj** *(cell: str, **kwargs) → bool*

**is_known_type** *(cell, is_quoted=False)*
CleverCSV

```python
is_nan(cell, **kwargs)
is_number(cell, **kwargs)
is_percentage(cell, **kwargs)
is_time(cell, **kwargs)
is_unicode_alphanum(cell, is_quoted=False, **kwargs)
is_unix_path(cell, **kwargs)
is_url(cell, **kwargs)
list_known_types()
clevercsv.detect_type.gen_known_type(cells)
Utility that yields a generator over whether or not the provided cells are of a known type or not.
clevercsv.detect_type.type_score(data, dialect, eps=1e-10)
Compute the type score as the ratio of cells with a known type.
Parameters
• data (str) – the data as a single string
• dialect (SimpleDialect) – the dialect to use
• eps (float) – the minimum value of the type score

clevercsv.dialect module
Definitions for the dialect object.
Author: Gertjan van den Burg
class clevercsv.dialect.SimpleDialect(delimiter, quotechar, escapechar, strict=False)
Bases: object
The simplified dialect object.
For the delimiter, quotechar, and escapechar the empty string means no delimiter/quotechar/escapechar in the file. None is used to mark it undefined.
Parameters
• delimiter (str) – The delimiter of the CSV file.
• quotechar (str) – The quotechar of the file.
• escapechar (str) – The escapechar of the file.
• strict (bool) – Whether strict parsing should be enforced. Same as in the csv module.
classmethod deserialize(obj)
Deserialize dialect from a JSON object
classmethod from_csv_dialect(d)
classmethod from_dict(d)
serialize()
Serialize dialect to a JSON object
to_csv_dialect()
to_dict()
```

5.8. CleverCSV API Documentation 31
validate()

clevercsv.dict_read_write module

DictReader and DictWriter.
This code is entirely copied from the Python csv module. The only exception is that it uses the reader and writer classes from our package.

Author: Gertjan van den Burg

class clevercsv.dict_read_write.DictReader(f, fieldnames=None, restkey=None, restval=None, dialect='excel', *args, **kwds)
    Bases: object
    property fieldnames

class clevercsv.dict_read_write.DictWriter(f, fieldnames, restval='', extrasaction='raise', dialect='excel', *args, **kwds)
    Bases: object
    writeheader()
    writerow(rowdict)
    writerows(rowdicts)

clevercsv.encoding module

Functionality to detect file encodings

Author: G.J.J. van den Burg License: See the LICENSE file

This file is part of CleverCSV.

clevercsv.encoding.get_encoding(filename, try_cchardet=True)
    Get the encoding of the file

This function uses the chardet package for detecting the encoding of a file.

Parameters

- **filename** (str) – Path to a file
- **try_cchardet** (bool) – Whether to run detection using cChardet if it is available. This can be faster, but may give different results than using chardet.

Returns **encoding** – Encoding of the file.

Return type  str
clevercsv.escape module

Common functions for dealing with escape characters.
Author: Gertjan van den Burg Date: 2018-11-06

clevercsv.escape.is_potential_escapechar(char, encoding, block_char=None)
Check if a character is a potential escape character.

A character is considered a potential escape character if it is in the “Punctuation, Other” Unicode category and in the list of blocked characters.

Parameters

- `char` (str) – The character to check
- `encoding` (str) – The encoding of the character
- `block_char` (iterable) – Characters that are in the Punctuation Other category but that should not be considered as escape character. If None, the default set is used, equal to:

```
['!', '?', ',', '.', ';', '(', ')', '|', '<', '>', '/', '\', '~', '@', '#', '%', '*', '&', '
```

Returns `is_escape` – Whether the character is considered a potential escape or not.

Return type bool

clevercsv.exceptions module

Exceptions for CleverCSV
Author: Gertjan van den Burg

exception clevercsv.exceptions.Error
Bases: cparsers.Error

exception clevercsv.exceptions.NoDetectionResult
Bases: Exception

clevercsv.normal_form module

Detect the dialect with very strict functional tests.
This module uses so-called “normal forms” to detect the dialect of CSV files. Normal forms are detected with strict functional tests. The normal forms are used as a pre-test to check if files are simple enough that computing the data consistency measure is not necessary.
Author: Gertjan van den Burg

clevercsv.normal_form.detect_dialect_normal(data, encoding='UTF-8', delimiters=None, verbose=False)
Detect the normal form of a file from a given sample

Parameters

- `data` (str) – The data as a single string
- `encoding` (str) – The encoding of the data

Returns `dialect` – The dialect detected using normal forms, or None if no such dialect can be found.

Return type SimpleDialect

clevercsv.normal_form.even_rows(rows, dialect)
clevercsv.normal_form.every_row_has_delim(rows, dialect)
clevercsv.normal_form.has_delimiter(string, delim)
clevercsv.normal_form.has_nested_quotes(string, quotechar)
clevercsv.normal_form.is_any_empty(cell)
clevercsv.normal_form.is_any_partial_quoted_cell(cell)
clevercsv.normal_form.is_any_quoted_cell(cell)
clevercsv.normal_form.is_elementary(cell)
clevercsv.normal_form.is_empty_quoted(cell, quotechar)
clevercsv.normal_form.is_empty_unquoted(cell)
clevercsv.normal_form.is_form_1(data, dialect=None)
clevercsv.normal_form.is_form_2(data, dialect)
clevercsv.normal_form.is_form_3(data, dialect)
clevercsv.normal_form.is_form_4(data, dialect)
clevercsv.normal_form.is_form_5(data, dialect)
clevercsv.normal_form.is_quoted_cell(cell, quotechar)
clevercsv.normal_form.maybe_has_escapechar(data, encoding, delim, quotechar)
clevercsv.normal_form.split_file(data)
clevercsv.normal_form.split_row(row, dialect)
clevercsv.normal_form.strip_trailing_crnl(data)

clevercsv.potential_dialects module

Code for selecting the potential dialects of a file.

Author: Gertjan van den Burg

clevercsv.potential_dialects.filter_urls(data)
    Filter URLs from the data

clevercsv.potential_dialects.get_delimiters(data, encoding, delimiters=None, block_cat=None, block_char=None)
    Get potential delimiters

    The set of potential delimiters is constructed as follows. For each unique character of the file, we check if its
    Unicode character category is in the set block_cat of prohibited categories. If it is, we don’t allow it to be
    a delimiter, with the exception of Tab (which is in the Control category). We furthermore block characters in
    block_char from being delimiters.

    Parameters
    
    • data (str) – The data of the file
    • encoding (str) – The encoding of the file
    • delimiters (iterable) – Allowed delimiters. If provided, it overrides the
      block_cat/block_char mechanism and only the provided characters will be considered delimiters (if they occur in the file). If None, all characters can be considered delimiters subject to
      the block_cat and block_char parameters.
• **block_cat** *(list)* – List of Unicode categories (2-letter abbreviations) for characters that should not be considered as delimiters. If None, the following default set is used:

```python
["Lu", "Ll", "Lt", "Lm", "Lo", "Nd", "Nl", "No", "Ps", "Pe", "Co"]
```

• **block_char** *(list)* – Explicit list of characters that should not be considered delimiters. If None, the following default set is used:

```python
[".", "/", ",", ",", ",", ",", ","]
```

**Returns** *delims* – Set of potential delimiters. The empty string is added by default.

**Return type** *set*

```python
clevercsv.potential_dialects.get_dialects(data, encoding='UTF-8', delimiters=None, test_masked_by_quotes=False)
```

Return the possible dialects for the given data.

We consider as escape characters those characters for which is_potential_escapechar() is True and that occur at least once before a quote character or delimiter in the dialect.

One may wonder if self-escaping is an issue here (i.e. “\”, two times backslash). It is not. In a file where a single backslash is desired and escaping with a backslash is used, then it only makes sense to do this in a file where the backslash is already used as an escape character (in which case we include it). If it is never used as escape for the delimiter or quotechar, then it is not necessary to self-escape. This is an assumption, but it holds in general and it reduces noise.

**Parameters**

- **data** *(str)* – The data for the file
- **encoding** *(str)* – The encoding of the file
- **delimiters** *(iterable)* – Set of delimiters to consider. See get_delimiters() for more info.
- **test_masked_by_quotes** *(bool)* – Remove dialects where the delimiter is always masked by the quote character. Enabling this typically removes a number of potential dialects from the list, which can remove false positives. It however not a very fast operation, so it is disabled by default.

**Returns** *dialects* – List of SimpleDialect objects that are considered potential dialects.

**Return type** *list*

```python
clevercsv.potential_dialects.get_quotechars(data, quote_chars=None)
```

Get potential quote characters

Quote characters are those that occur in the quote_chars set and are found at least once in the file.

**Parameters**

- **data** *(str)* – The data of the file as a string
- **quote_chars** *(iterable)* – Characters that should be considered quote characters. If it is None, the following default set is used:

```python
["\n", "\r", "\", "\", "\", "\", "\", "\", "\"]
```

**Returns** *quotes* – Set of potential quote characters. The empty string is added by default.

**Return type** *set*
CleverCSV

clevercsv.potential_dialects.masked_by_quotechar(data, quotechar, escapechar, test_char)

Test if a character is always masked by quote characters

This function tests if a given character is always within quoted segments (defined by the quote character). Double quoting and escaping is supported.

Parameters
- data (str) – The data of the file as a string
- quotechar (str) – The quote character
- escapechar (str) – The escape character
- test_char (str) – The character to test

Returns masked – Returns True if the test character is never outside quoted segments, False otherwise.

Return type bool

clevercsv.potential_dialects.unicode_category(x, encoding=None)

Return the Unicode category of a character

Parameters
- x (str) – character
- encoding (str) – Encoding of the character

Returns category – The Unicode category of the character.

Return type str

clevercsv.read module

Drop-in replacement for the Python csv reader class. This is a wrapper for the Parser class, defined in cparser.

Author: Gertjan van den Burg

class clevercsv.read.reader(csvfile, dialect='excel', **fmtparams)

   Bases: object

   next()

clevercsv.utils module

Various utilities

Author: Gertjan van den Burg

clevercsv.utils.pairwise(iterable)
   s - > (s0, s1), (s1, s2), (s2, s3), ...

clevercsv.utils.sha1sum(filename)
   Compute the SHA1 checksum of a given file

   Parameters filename (str) – Path to a file

   Returns checksum – The SHA1 checksum of the file contents.

   Return type str
**clevercsv.wrappers module**

Wrappers for some loading/saving functionality.

Author: Gertjan van den Burg

```python

clevercsv.wrappers.detect_dialect(filename, num_chars=None, encoding=None, verbose=False, method='auto', skip=True)
```

Detect the dialect of a CSV file

This is a utility function that simply returns the detected dialect of a given CSV file.

**Parameters**

- `filename` (str) – The filename of the CSV file.
- `num_chars` (int) – Number of characters to read for the detection. If None, the entire file will be read. Note that limiting the number of characters can reduce the accuracy of the detected dialect.
- `encoding` (str) – The file encoding of the CSV file. If None, it is detected.
- `verbose` (bool) – Enable verbose mode during detection.
- `method` (str) – Dialect detection method to use. Either ‘normal’ for normal form detection, ‘consistency’ for the consistency measure, or ‘auto’ for first normal and then consistency.
- `skip` (bool) – Skip computation of the type score for dialects with a low pattern score.

**Returns**

`dialect` – The detected dialect as a SimpleDialect, or None if detection failed.

**Return type**

SimpleDialect

```python

clevercsv.wrappers.read_dataframe(filename, *args, num_chars=None, **kwargs)
```

Read a CSV file to a Pandas dataframe

This function uses CleverCSV to detect the dialect, and then passes this to the `read_csv` function in pandas. Additional arguments and keyword arguments are passed to `read_csv` as well.

**Parameters**

- `filename` (str) – Path of the CSV file
- `*args` – Additional arguments for the `pandas.read_csv` function.
- `num_chars` (int) – Number of characters to use for dialect detection. If None, use the entire file.

Note that using less than the entire file will speed up detection, but can reduce the accuracy of the detected dialect.

- `**kwargs` – Additional keyword arguments for the `pandas.read_csv` function. You can specify the file encoding here if needed, and it will be used during dialect detection.

```python

clevercsv.wrappers.read_dicts(filename, dialect=None, encoding=None, num_chars=None, verbose=False)
```

Read a CSV file as a list of dictionaries

This function returns the rows of the CSV file as a list of dictionaries. The keys of the dictionaries are assumed to be in the first row of the CSV file. The dialect will be detected automatically, unless it is provided.

**Parameters**

- `filename` (str) – Path of the CSV file
• **dialect** (str, SimpleDialect, or csv.Dialect object) – If the dialect is known, it can be provided here. This function uses the Clevercsv clevercsv.DictReader object, which supports various dialect types (string, SimpleDialect, or csv.Dialect). If None, the dialect will be detected.

• **encoding** (str) – The encoding of the file. If None, it is detected.

• **num_chars** (int) – Number of characters to use to detect the dialect. If None, use the entire file.

  Note that using less than the entire file will speed up detection, but can reduce the accuracy of the detected dialect.

• **verbose** (bool) – Whether or not to show detection progress.

  **Returns** rows – Returns rows of the file as a list of dictionaries.

  **Return type** list

  **Raises** NoDetectionResult – When the dialect detection fails.

**clevercsv.wrappers.read_table(filename, dialect=None, encoding=None, num_chars=None, verbose=False)**

Read a CSV file as a table (a list of lists)

This is a convenience function that reads a CSV file and returns the data as a list of lists (= rows). The dialect will be detected automatically, unless it is provided.

**Parameters**

• **filename** (str) – Path of the CSV file

• **dialect** (str, SimpleDialect, or csv.Dialect object) – If the dialect is known, it can be provided here. This function uses the CleverCSV clevercsv.reader object, which supports various dialect types (string, SimpleDialect, or csv.Dialect). If None, the dialect will be detected.

• **encoding** (str) – The encoding of the file. If None, it is detected.

• **num_chars** (int) – Number of characters to use to detect the dialect. If None, use the entire file.

  Note that using less than the entire file will speed up detection, but can reduce the accuracy of the detected dialect.

• **verbose** (bool) – Whether or not to show detection progress.

  **Returns** rows – Returns rows as a list of lists.

  **Return type** list

  **Raises** NoDetectionResult – When the dialect detection fails.

**clevercsv.wrappers.stream_dicts(filename, dialect=None, encoding=None, num_chars=None, verbose=False)**

Read a CSV file as a generator over dictionaries

This function streams the rows of the CSV file as dictionaries. The keys of the dictionaries are assumed to be in the first row of the CSV file. The dialect will be detected automatically, unless it is provided.

**Parameters**

• **filename** (str) – Path of the CSV file

• **dialect** (str, SimpleDialect, or csv.Dialect object) – If the dialect is known, it can be provided here. This function uses the CleverCSV clevercsv.DictReader object,
CleverCSV

which supports various dialect types (string, SimpleDialect, or csv.Dialect). If None, the
dialect will be detected.

• **encoding** *(str)* – The encoding of the file. If None, it is detected.

• **num_chars** *(int)* – Number of characters to use to detect the dialect. If None, use the entire
file.

  Note that using less than the entire file will speed up detection, but can reduce the accuracy
  of the detected dialect.

• **verbose** *(bool)* – Whether or not to show detection progress.

  **Returns** rows – Returns file as a generator over rows as dictionaries.

  **Return type** generator

  **Raises** NoDetectionResult – When the dialect detection fails.

**clevercsv.wrappers.stream_table** *(filename, dialect=None, encoding=None, num_chars=None, verbose=False)*

Read a CSV file as a generator over rows of a table

This is a convenience function that reads a CSV file and returns the data as a generator of rows. The dialect will
be detected automatically, unless it is provided.

**Parameters**

• **filename** *(str)* – Path of the CSV file

• **dialect** *(str, SimpleDialect, or csv.Dialect)* – If the dialect is known,
it can be provided here. This function uses the CleverCSV clevercsv.reader object,
which supports various dialect types (string, SimpleDialect, or csv.Dialect). If None, the
dialect will be detected.

• **encoding** *(str)* – The encoding of the file. If None, it is detected.

• **num_chars** *(int)* – Number of characters to use to detect the dialect. If None, use the entire
file.

  Note that using less than the entire file will speed up detection, but can reduce the accuracy
  of the detected dialect.

• **verbose** *(bool)* – Whether or not to show detection progress.

  **Returns** rows – Returns file as a generator over rows.

  **Return type** generator

  **Raises** NoDetectionResult – When the dialect detection fails.

**clevercsv.wrappers.write_table** *(table, filename, dialect='excel', transpose=False, encoding=None)*

Write a table (a list of lists) to a file

This is a convenience function for writing a table to a CSV file.

**Parameters**

• **table** *(list)* – A table as a list of lists. The table must have the same number of cells in
each row (taking the transpose flag into account).

• **filename** *(str)* – The filename of the CSV file to write the table to.

• **dialect** *(SimpleDialect or csv.Dialect)* – The dialect to use. The default is the ’excel’ dialect, which corresponds to RFC4180. This is done to encourage more standardized CSV files.
• **transpose** *(bool)* – Transpose the table before writing.

• **encoding** *(str)* – Encoding to use to write the data to the file. Note that the default encoding is platform dependent, which ensures compatibility with the Python open() function. It thus defaults to `locale.getpreferredencoding()`.

**Raises** **ValueError**: – When the length of the rows is not constant.

clevercsv.write module

Drop-in replacement for the Python csv writer class.

Author: Gertjan van den Burg

class clevercsv.write.writer(*csvfile, dialect='excel', **fmtparams*)
    Bases: object
    writerow(*row*)
    writerows(*rows*)

Module contents

• genindex
• modindex
• search
C

clevercsv, 40
clevercsv.break_ties, 26
clevercsv.consistency, 28
clevercsv.cparser_util, 28
clevercsv.detect, 29
clevercsv.detect_pattern, 29
clevercsv.detect_type, 30
clevercsv.dialect, 31
clevercsv.dict_read_write, 32
clevercsv.encoding, 32
clevercsv.escape, 33
clevercsv.exceptions, 33
clevercsv.normal_form, 33
clevercsv.potential_dialects, 34
clevercsv.read, 36
clevercsv.utils, 36
clevercsv.wrappers, 37
clevercsv.write, 40
INDEX

B
break_ties() (in module clevercsv.consistency), 28
break_ties_four() (in module clevercsv.break_ties), 26
break_ties_three() (in module clevercsv.break_ties), 26
break_ties_two() (in module clevercsv.break_ties), 27

C
clevercsv module, 40
clevercsv.break_ties module, 26
clevercsv.consistency module, 28
clevercsv.cparser_util module, 28
clevercsv.detect module, 29
clevercsv.detect_pattern module, 29
clevercsv.detect_type module, 30
clevercsv.dialect module, 31
clevercsv.dict_read_write module, 32
clevercsv.encoding module, 32
clevercsv.escape module, 33
clevercsv.exceptions module, 33
clevercsv.normal_form module, 33
clevercsv.potential_dialects module, 34
clevercsv.read module, 36
clevercsv.utils module, 36
clevercsv.wrappers module, 37
clevercsv.write module, 40
consistency_scores() (in module clevercsv.consistency), 28

deserialize() (clevercsv.dialect.SimpleDialect class method), 31
detect() (clevercsv.detect.Detector method), 29
detect_consistency_dialects() (in module clevercsv.consistency), 28
detect_dialect() (in module clevercsv.wrappers), 37
detect_dialect_consistency() (in module clevercsv.consistency), 28
detect_dialect_normal() (in module clevercsv.normal_form), 33
detect_type() (clevercsv.detect_type.TypeDetector method), 30
Detector (class in clevercsv.detect), 29
DictReader (class in clevercsv.dict_read_write), 32
DictWriter (class in clevercsv.dict_read_write), 32

E
Error, 33
even_rows() (in module clevercsv.normal_form), 33
every_row_has_delim() (in module clevercsv.normal_form), 34

F
field_size_limit() (in module clevercsv.cparser_util), 28
fieldnames (clevercsv.dict_read_write.DictReader property), 32
fill_empty() (in module clevercsv.detect_pattern), 29
filter_urls() (in module clevercsv.potential_dialects), 34
from_csv_dialect() (clevercsv.dialect.SimpleDialect class method), 31
from_dict() (clevercsv.dialect.SimpleDialect class method), 31
is_nan() (clevercsv.detect_type.TypeDetector method), 31
is_number() (clevercsv.detect_type.TypeDetector method), 31
is_percentage() (clevercsv.detect_type.TypeDetector method), 31
is_potential_escapechar() (in module clevercsv.escape), 33
is_quoted_cell() (in module clevercsv.normal_form), 34
is_time() (clevercsv.detect_type.TypeDetector method), 31
is_unicode_alphanum() (clevercsv.detect_type.TypeDetector method), 31
is_url() (clevercsv.detect_type.TypeDetector method), 31
list_known_types() (clevercsv.detect_type.TypeDetector method), 31
make_abstraction() (in module clevercsv.detect_pattern), 29
masked_by_quotechar() (in module clevercsv.potential_dialects), 35
maybe_has_escapechar() (in module clevercsv.normal_form), 34
merge_with_quotechar() (in module clevercsv.detect_pattern), 30
module
clevercsv, 40
clevercsv.break_ties, 26
clevercsv.consistency, 28
clevercsv.cparser_util, 28
clevercsv.detect, 29
clevercsv.detect_pattern, 29
clevercsv.detect_type, 30
clevercsv.dialect, 31
clevercsv.dict_read_write, 32
clevercsv.encoding, 32
clevercsv.escape, 33
clevercsv.exceptions, 33
clevercsv.normal_form, 33
clevercsv.potential_dialects, 34
clevercsv.read, 36
clevercsv.utils, 36
clevercsv.wrappers, 37
clevercsv.write, 40
Index 45