

CSV Filter

We will write a program called `csvfilter.py` that will filter a delimited text file for some value, possibly in one of the columns.

The program should take the following parameters:

- `-f` | `--file`: A **required** argument that is a readable file
- `-v` | `--val`: A **required** "value" to match against each record
- `-c` | `--col`: An optional "column" to search for the given value
- `-o` | `--outfile`: An optional output file name (default `'out.csv'`)
- `-d` | `--delimiter`: An optional delimiter to use to parse the file (default `','`)

Here is the expected usage for the program:

```
$ ./csvfilter.py -h
usage: csvfilter.py [-h] -f FILE -v val [-c col] [-o OUTFILE] [-d delim]

Filter delimited records

optional arguments:
  -h, --help            show this help message and exit
  -f FILE, --file FILE  Input file (default: None)
  -v val, --val val     Value for filter (default: None)
  -c col, --col col     Column for filter (default: )
  -o OUTFILE, --outfile OUTFILE
                        Output filename (default: out.csv)
  -d delim, --delimiter delim
                        Input delimiter (default: ,)
```

The program will search the `--file` for the `--val` value either in the given `--col` column or anywhere on the line in a *case-insensitive* fashion. Any records matching will be written to the `--outfile`. The input files are delimited by commas and tabs, so you will need to use the `--delimiter` option to parse them.

Titanic

For the following, I will use the `csvchk` program to inspect the first record of the input files. You can install this like so:

```
$ python3 -m pip install csvchk
```

Let's take a look at the `titanic.csv` file:

```
$ csvchk inputs/titanic.csv
// ***** Record 1 ***** //
id      : 0
survived : 0
pclass  : 3
sex      : male
age      : 22.0
sibsp    : 1
parch    : 0
fare     : 7.25
embarked : S
class    : Third
who      : man
adult_male : True
deck     :
embark_town : Southampton
alive    : no
alone    : False
```

If wanted to find how many records have the value "true" (case-insensitive), I could use **grep**:

```
$ grep -i true inputs/titanic.csv | wc -l
664
```

So if I run my program to search for this value, I should get the same number:

```
$ ./csvfilter.py -v true -f inputs/titanic.csv
Done, wrote 664 to "out.csv".
```

Since the output also has a header line, the number of lines in the **out.csv** file should be 665:

```
$ wc -l out.csv
665 out.csv
```

This type of search is looking for the string "true" (case-insensitive) *anywhere on the line*. If I wanted to only look in the **adult_male** column, however, I would get a different number:

```
$ ./csvfilter.py -v true -c adult_male -f inputs/titanic.csv
Done, wrote 537 to "out.csv".
```

Let's verify that. The **csvchk** has an option **-n** to *number* the columns:

```
$ csvchk -n inputs/titanic.csv
// ***** Record 1 ***** //
 1 id      : 0
 2 survived : 0
 3 pclass  : 3
 4 sex     : male
 5 age     : 22.0
 6 sibsp   : 1
 7 parch   : 0
 8 fare    : 7.25
 9 embarked : S
10 class   : Third
11 who     : man
12 adult_male : True
13 deck    :
14 embark_town : Southampton
15 alive   : no
16 alone   : False
```

I see the `adult_male` column is 12, so I can use that with `awk` to extract the 12th column and `grep` for "true":

```
$ awk -F',' '{print $12}' inputs/titanic.csv | grep -i true | wc -l
537
```

OK, that looks legit.

Note that the output file has all the input columns and is in CSV format:

```
$ csvchk out.csv
// ***** Record 1 ***** //
id      : 0
survived : 0
pclass  : 3
sex     : male
age     : 22.0
sibsp   : 1
parch   : 0
fare    : 7.25
embarked : S
class   : Third
who     : man
adult_male : True
deck    :
embark_town : Southampton
alive   : no
alone   : False
```

Centroids

Let's now check out the `centroids.txt` file:

```
$ csvchk inputs/centroids.tab
// ***** Record 1 ***** //
centroid : e5d49c0803f04032b482a1ee836e18ab
domain   : Bacteria
kingdom   : Proteobacteria
phylum  : Alphaproteobacteria
class     : Rhodospirillales
order     : AEGEAN-169 marine group
genus     : uncultured bacterium
species   : uncultured bacterium
```

The string "bacteria" occurs on 493 lines:

```
$ grep -i bacteria inputs/centroids.tab | wc -l
493
```

To parse this file, we'll need to indicate that the `--delimiter` is a tab character:

```
$ ./csvfilter.py -d '$\t' -v BACTERIA -f inputs/centroids.tab -o bacteria.csv
Done, wrote 493 to "bacteria.csv".
```

Ensure that the output file actually has the correct number of records:

```
$ wc -l bacteria.csv
494 bacteria.csv
```

If, however, we limit our search to the "class" column, the string "bacteria" occurs only 50 times:

```
$ ./csvfilter.py -d '$\t' -v BACTERIA -f inputs/centroids.tab -o bacteria.csv -c class
Done, wrote 50 to "bacteria.csv".
$ wc -l bacteria.csv
51 bacteria.csv
```

Parsing and writing delimited files

For this exercise, you will need to use the `csv` module, specifically:

- `csv.DictReader`
- `csv.DictWriter`

Be sure to read the [documentation](#)!

Your `args.file` should be an open file handle, so you can create a reader:

```
reader = csv.DictReader(args.file, delimiter=args.delimiter)
```

This object has a `fieldnames` attribute which you should use to verify that the given `--col` argument is actually a valid option. If it is not, your program should print an error message (preferably to `STDERR`) and exit *with an error value*. You may optionally display the valid fieldnames, but this is not tested:

```
$ ./csvfilter.py -d $'\t' -v BACTERIA -f inputs/centroids.tab -c clas
--col "clas" not a valid column!
Choose from centroid, domain, kingdom, phylum, class, order, genus, species
```

You can use the `csv.DictWriter` to create a writer which can be used to write the header row to the output file:

```
writer = csv.DictWriter(args.outfile, fieldnames=reader.fieldnames)
writer.writeheader()
```

You can use a `for` loop to iterate through each record in the input file where the record will be represented as a `dict` having the first row column headers for the keys and the current record's data as the values. Try this to start:

```
for rec in reader:
    print(rec)
    break
```

You will search for the indicated `--val` in all the `rec.values()` or just the given `--col` column from the record. I would suggest you use a regular expression with the case-insensitive option:

```
if re.search(search_for, text, re.IGNORECASE):
    # write the output
```

You can refer to the `csv.DictWriter` documentation to see how to use the `writer` to write the record in a way that is similar to how we have used `SeqIO.write()` to write a sequence record to an output file handle.

When you are done, be sure to let the user know how many records were written to which output file name.

Tests

A passing test suite:

```
$ make test
pytest -xv --disable-pytest-warnings test.py
===== test session starts =====
...

test.py::test_exists PASSED [ 12%]
test.py::test_usage PASSED [ 25%]
test.py::test_bad_file PASSED [ 37%]
test.py::test_bad_column PASSED [ 50%]
test.py::test_titanic_any_true PASSED [ 62%]
test.py::test_titanic_adult_male_true PASSED [ 75%]
test.py::test_centroids_any_bacteria PASSED [ 87%]
test.py::test_centroids_class_bacteria PASSED [100%]

===== 8 passed in 0.51s =====
```