**Running PeakFilter user guide**

**Purpose**

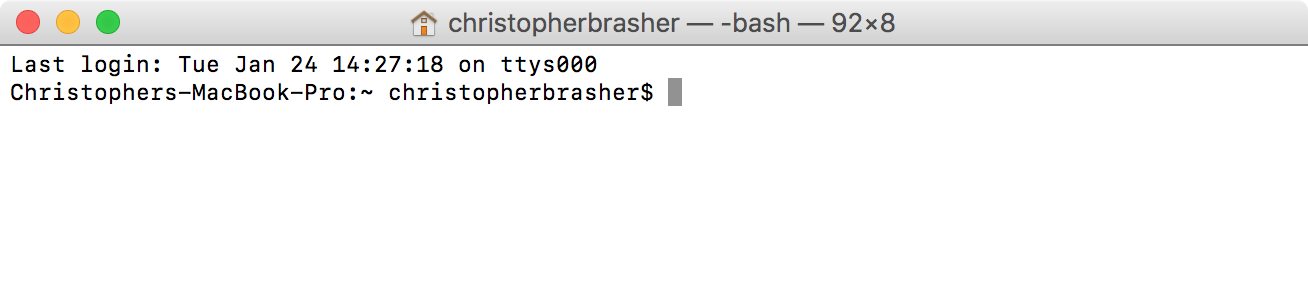
This guide details how to process prepared data files with Python.

**Prerequisites**

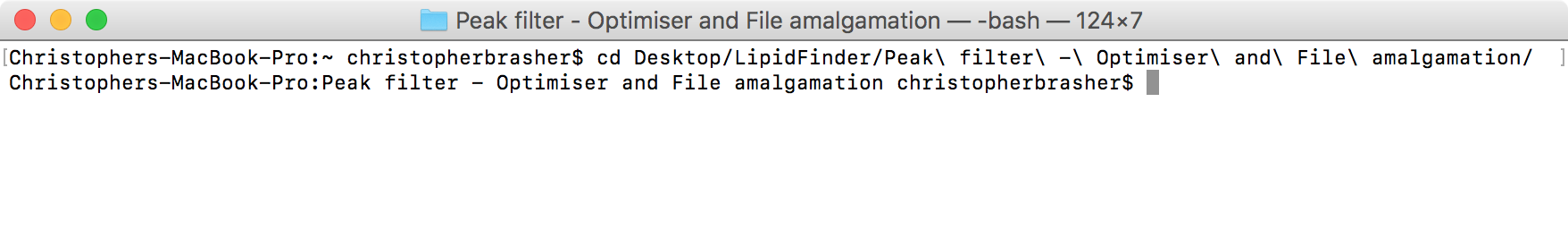
1. Python 2.7 minimum with specific libraries installed as per Python installation guide.
2. PeakFilter Input file(s) - LC/MS data that has been pre-processed with SIEVE and prepared as per PeakFilter\_Input\_file\_preparation\_guide and stored as a comma separated file(s) (csv)

**Processing file(s) with PeakFilter using Mac OSX/Linux**

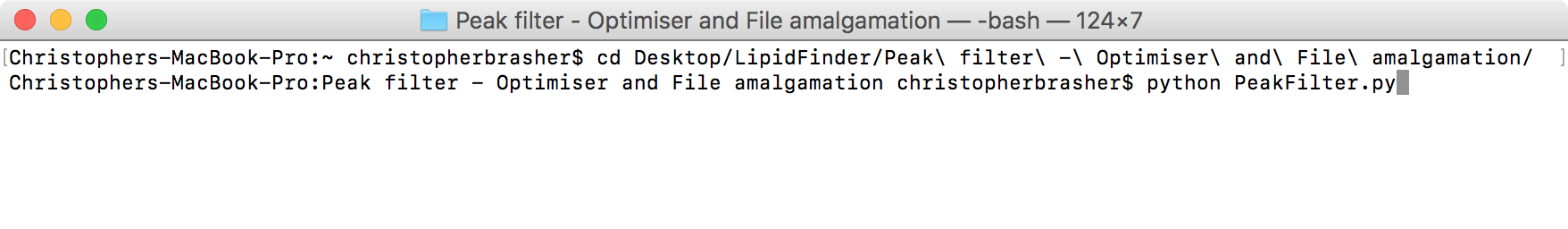
1. Open Terminal



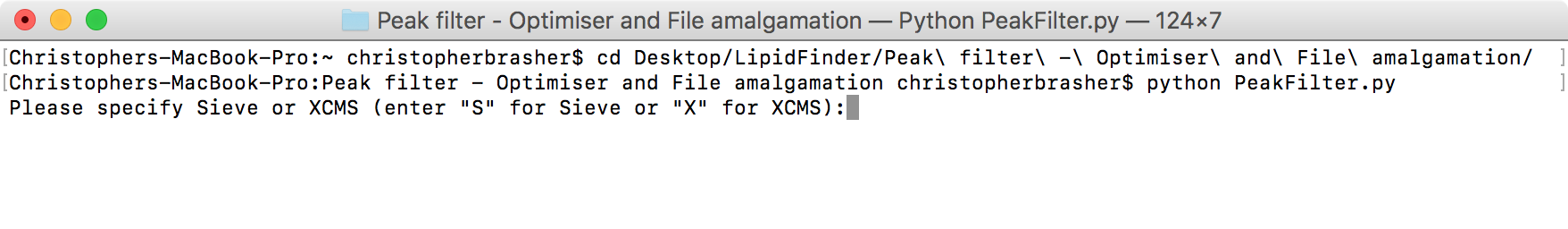
1. Navigate to the folder where the PeakFilter code is.



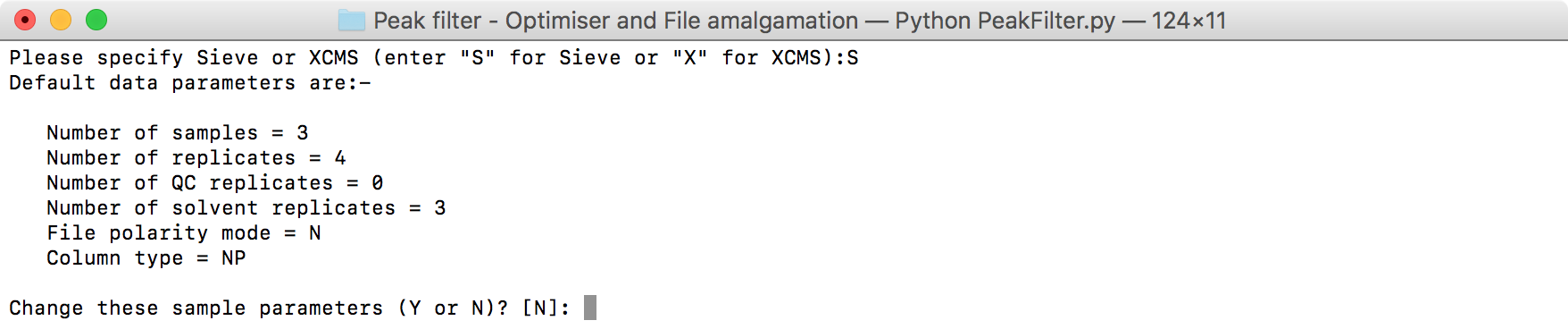
1. Type in the console – “python PeakFilter.py” (this is case sensitive) – This will start the PeakFilter program and import all dependant libraries.



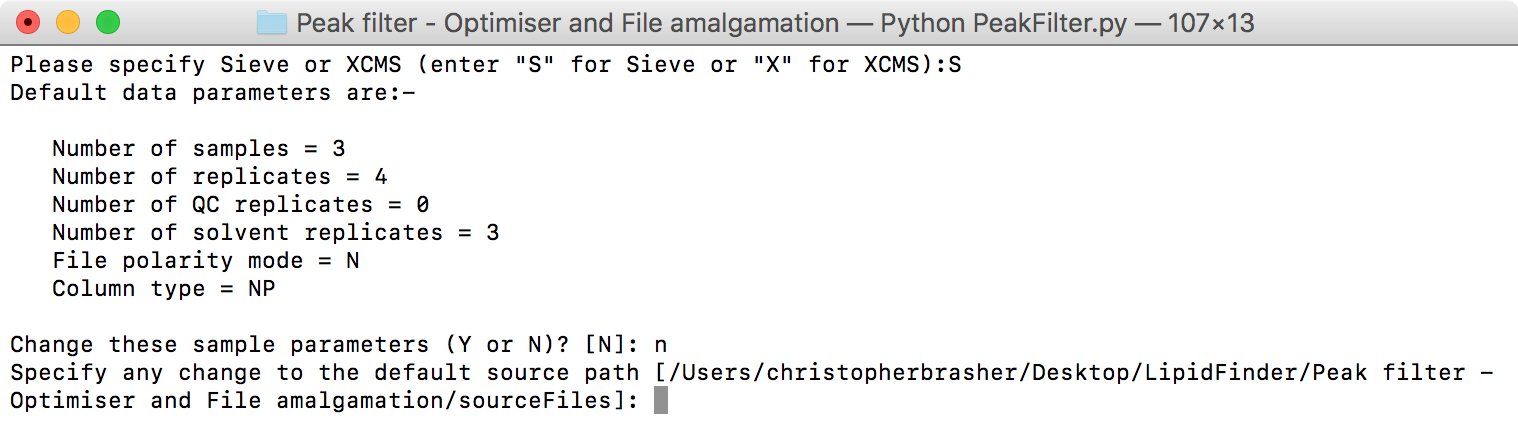
1. PeakFilter will prompt for the type of input file ‘SIEVE’ or ‘XCMS’.



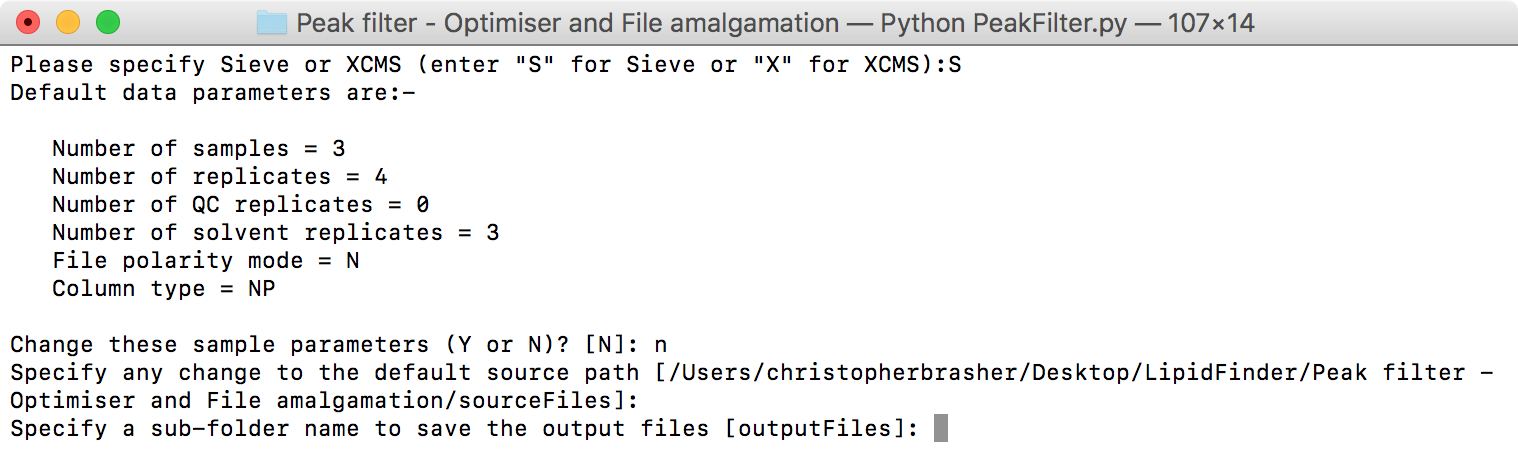
1. PeakFilter will prompt for confirmation of the input file details.
   1. If these have been modified in the parameters.csv file in the PeakFilter folder before step 5 then these should be correct. However, these can be modified at this point by entering “Y” at the prompt. Each file parameter will now be prompted for individually.
   2. If the parameters are correct accept the default ‘No’ (‘[N]’) and hit return.



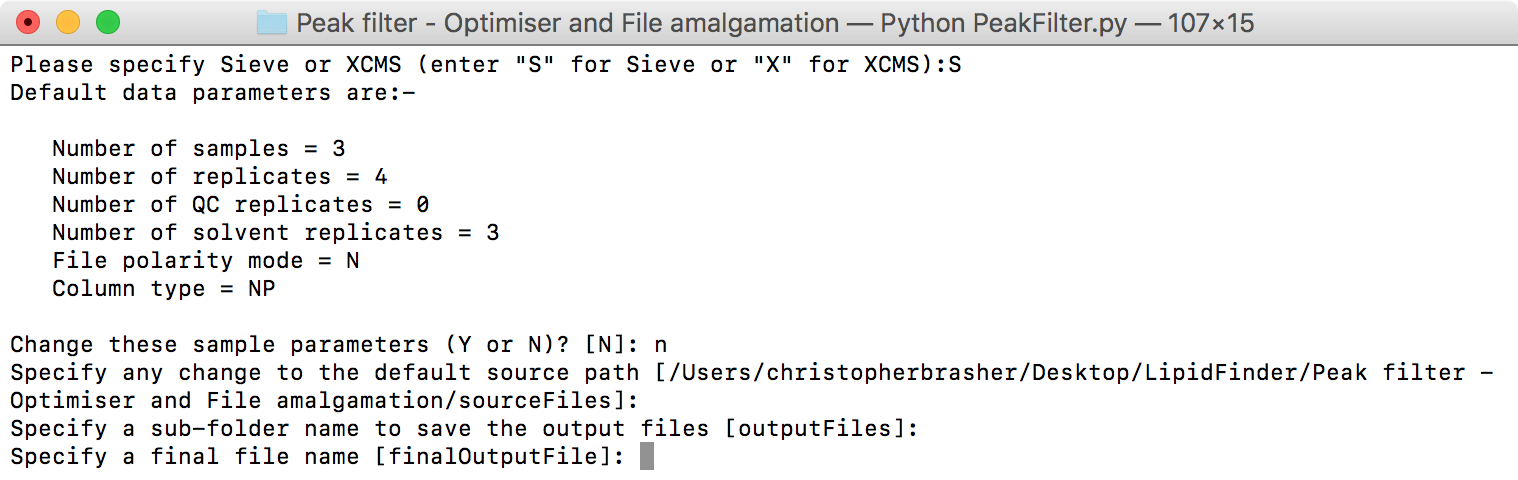
1. PeakFilter input files can read from anywhere on the computer file system, however the default location is ‘PeakFilter/sourceFiles’. If there are multiple input files they must reside in the same folder. To change the default path insert the full path to the input file folder here.



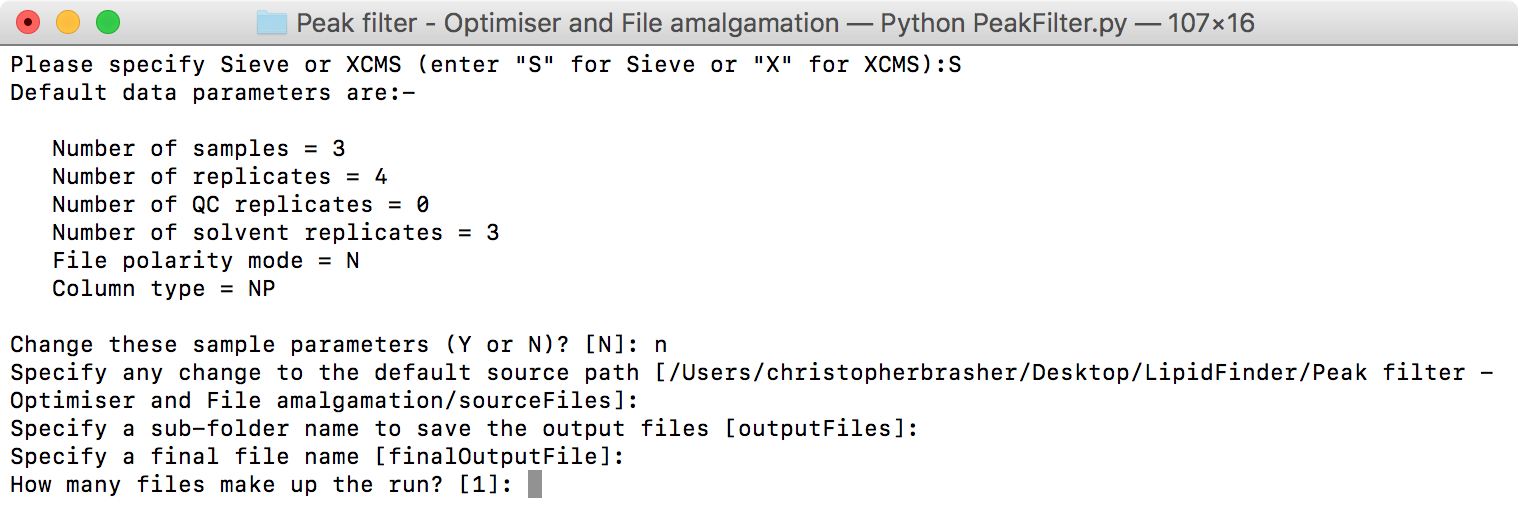
1. An output file is created at each stage during the PeakFilter affording a comprehensive audit trail. These files, by default are stored in a sub folder of the PeakFilter folder in a folder named ‘outputFiles’ (PeakFilter/outputFiles). Specify here a different sub-folder to save the PeakFilter output, if it does not exist it will be created provided the naming is legal. It is recommended to use a different output folder for each PeakFilter run since up to 18 files can be generated for each PeakFilter run.



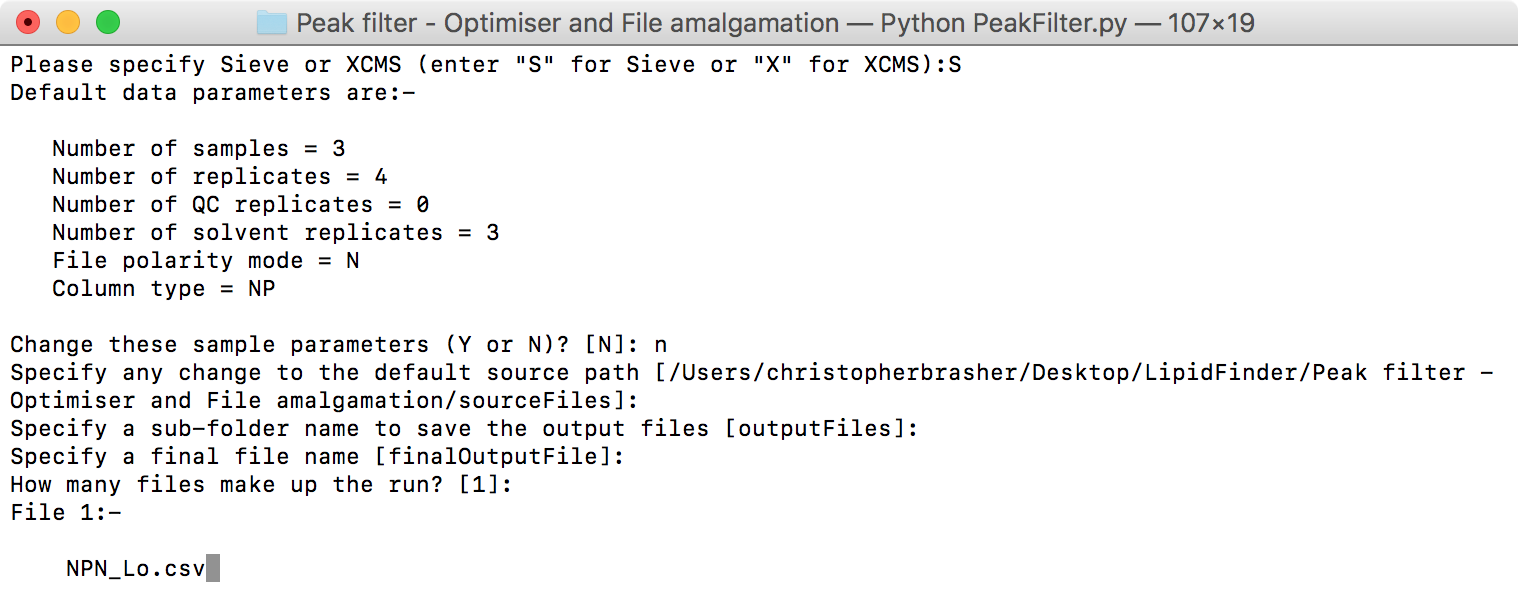
1. By default, the final two files generated during the PeakFilter process (the summary file consisting of mean sample intensities, retention time and m/z; and the details file, which contains the final version of the all the data as processed) are prefixed with the name ‘finalOutputfFile’. At this point the user has the option to change this default prefix to something more meaningful.



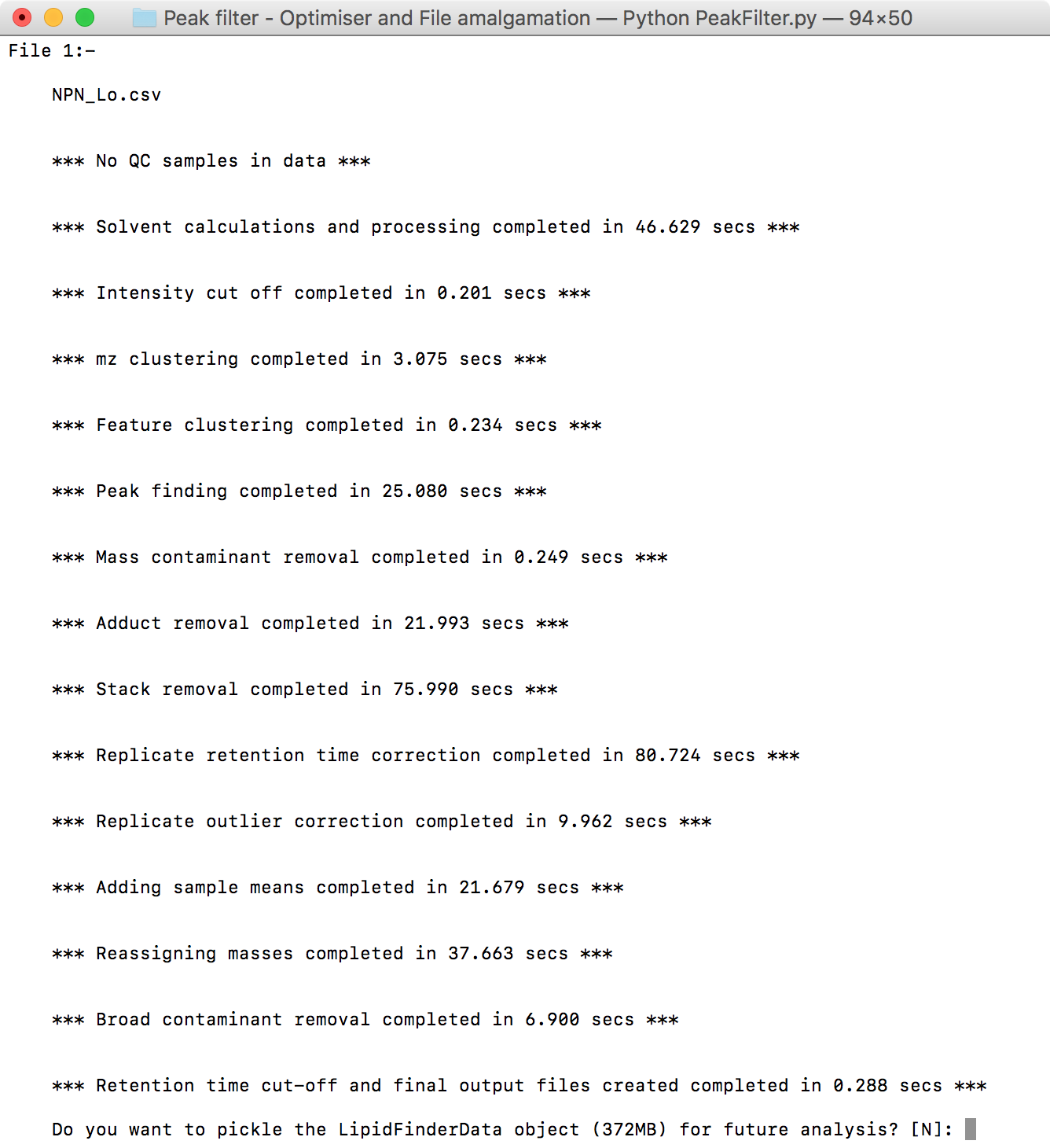
1. Occasionally, there may be more than one input file if the SIEVE runs have been split by retention time ranges to ease the processing requirements on the computer. At this point the default 1 file input file can be changed to any number of files provided the rules described in the ‘PeakFilter input file preparation guide’ are followed. These can be found in the ‘File formatting considerations’ section.



1. PeakFilter will now prompt for each of the input files in turn, input files should be loaded in the order smallest retention time to largest retention time.



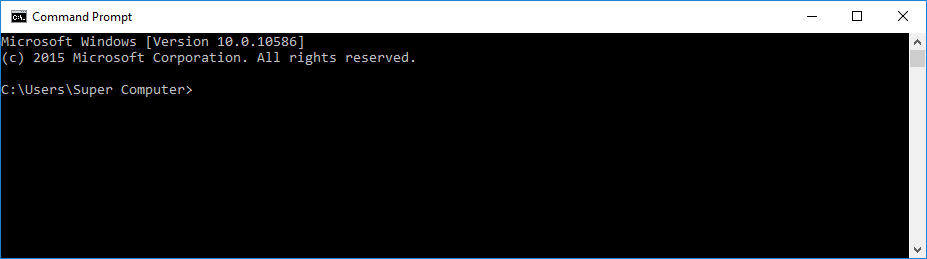
1. After the last input file is successfully loaded, the PeakFilter process will execute to completion based on the input parameters. In larger datasets this can take some time, progress is reported at the end of each processing step. Additionally, at each step of the current dataset is saved in the output folder specified in step 9 to give full line item traceability. These datasets are saved as csv and and their names and contents are summarised in The PeakFilter workflow overview below. Once processing has completed an option is given to pickle the final state of the PeakFilterData object, this gives users who are familiar with Python the opportunity to examine the underlying data objects at each stage or to perform ‘what if?’ analysis on any of the stages of PeakFilter by adjusting parameters etc. Note that the PeakFilterData object contains a representation of each dataset produced during processing and as such requires a lot of memory to store.



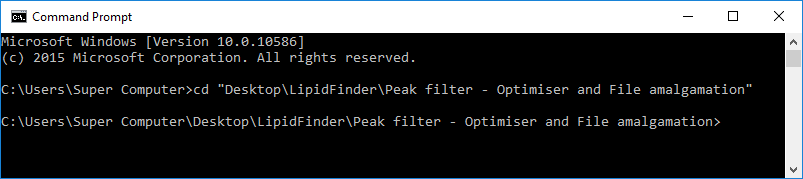
1. PeakFilter has completed.

**Running PeakFilter using Windows**

1. Open Command prompt.



1. Navigate to the folder where the PeakFilter code is.



1. Type in the console – “python PeakFilter.py” (this is case sensitive) – This will start the PeakFilter program and import all dependant libraries. Follow the instructions from section 4 under ‘Processing file(s) with PeakFilter using Mac OSX/Linux’ above.

**The PeakFilter workflow overview**

Output file naming

The name of each output file is made up of 2 parts, a prefix, which identifies the stage of the process that generated the file and a common suffix indicating the column type, polarity, year, month, date, hour, minute and second. For example:

2-QC\_data\_NP\_N\_20151207-160538.csv

(stage prefix) (column type, polarity) (year, month, date) (hour, minute, second)

Step 1 – Raw data importation

**Optional**: No

**Module**: PeakFilterData.py

**Description**: The input files are imported into 1 internal dataframe. If there is more than 1 input file splitting the retention time range into smaller chunks then the import will follow the rules from ‘PeakFilter input file preparation’, section ‘File formatting considerations’. An additional column is added to identify where the data in each frame originally came from.

**Stage prefix**: ‘1-Combined\_raw\_data’

Step 2 – QC Sample Calculations and Reporting

**Optional**: Not if QC samples are present

**Module**: qcCalcs.py

**Description**: For each frame the mean and the relative standard deviation (RSD) are calculated and recorded in new columns. The percentage of RSDs that are below the *lowerRSDCutOff* and the *upperRSDCutOff* are recorded. A percentage of *lowerRSDCutOff: upperRSDCutOff* is calculated and reported, the higher the percentage the lower technical variation between the QC samples overall.

**Stage prefix**: ‘2-QC\_data’

Step 3 – Solvent removal

**Optional**: Yes (even if solvent samples are present in dataset)

**Module**: solventCalcs.py

**Description**: The solvent sample replicates for each frame undergo outlier correction (outlierCorrect.py). The mean intensity of the remaining solvent replicates for each frame is calculated and stored in a new column. Frames where every replicate of every sample has an intensity that is not at least *solventFoldCutOff* \* the corresponding mean solvent intensity will be deleted from the dataset. Remaining sample replicates will have the mean solvent intensity removed from their intensity figure, with the minimum intensity being 0.

**Stage prefix**: ‘3-Solvent\_Data’

Step 4 – Low intensity removal

**Optional**: No

**Module**: solventCalcs.py

**Description**: Any replicate intensities less than *intensitySignificanceCutOff* are considered too insignificant to process and are set to 0. Frames where all replicates for all samples are 0 are deleted.

**Stage prefix**: ‘4-Low\_intensities\_removed’

Step 5 – m/z clustering

**Optional**: No

**Module**: clustering.py

**Description**: 2 stage hierarchical clustering is used to first section the data into groups of at least 50 frames. Each group undergoes hierarchical clustering to group frames with very similar m/z together to be treated as isobaric. A column is added to the dataset indicating each frame’s m/z cluster id.

**Stage prefix**: ‘5-Mass\_clustered\_data’

Step 6 – Feature set clustering

**Optional**: No

**Module**: clustering.py

**Description**: Frames within each mass cluster are sorted by retention time and contiguous frames, separated by a small enough retention time, are grouped and regarded as the same feature set.

**Stage prefix**: ‘6-Feature\_clustered\_data’

Step 7 – Feature peak analysis

**Optional**: No

**Module**: peakFinder.py

**Description**: Each feature set is examined individually to check for the existence of sharp, narrow peaks. Wider, flatter features are discarded as contamination.

**Stage prefix**: ‘7-Peak\_found\_data’

Step 8 – Mass contaminant removal

**Optional**: No, but list can be amended

**Module**: contaminantRemoval.py

**Description**: Frames are removed from the dataset that match (within a tolerance) the list of contaminant masses found in the file ‘contaminants.csv’. This is to allow contamination from known plasticisers or other fixed contaminants to be removed from the data. Retention time is not considered, if the mass is found at all it is removed.

**Stage prefix**: ‘8-Mass\_contaminant\_removed\_data’

Step 9 – Adduct ion removal

**Optional**: Yes

**Module**: contaminantRemoval.py

**Description**: A list of adducts and their mass differences for both positive and negative mode are maintained in the file *‘adducts.csv’*. A list of adduct ion pairings is kept in the file ‘parameters.csv’. Together these are used to check for the existence of adducts of each frame. Where an adduct is found and the retention time matches the parent ion the largest intensity m/z is retained and the lower intensity m/z is set to 0.

**Stage prefix**: ‘9-Adducts\_removed\_data’

Step 10 – Stack removal

**Optional**: No, but list can be amended

**Module**: contaminantRemoval.py

**Description**: Both lipid and contaminant stacks are removed. A list of mass multiples for both types of stacks is stored in the file *‘stacks.csv’*. Where a lipid stack is found all member frame of the stack are removed except the parent, in the case of a contaminant stack being found every member is removed.

**Stage prefix**: ‘10-Stacks\_removed\_data’

Step 11 – Replicate retention time correction

**Optional**: No, but list can be amended

**Module**: rtCorrect.py

**Description**: Following Sieve and the peak finding process retention time can be misaligned for some features in some replicates. Intensities are moved to nearby frames in cases where an intensity is in a sparsely populated frame would be better placed in adjacent gap in a densely populated frame.

**Stage prefix**: ‘11-Replicate\_Retention\_time\_corrected\_data’

Step 12 – Outlier correction

**Optional**: No, but parameters can be amended

**Module**: outlierCorrect.py

**Description**: Replicates within a sample for each frame are checked to ensure the variation between intensities is low. If variation is high an there are outliers, these can be removed in certain circumstances to reduce variation, otherwise all the replicates are set to zero as the sample data for the frame is unreliable.

**Stage prefix**: ‘12-Outlier\_corrected\_data’

Step 13 – Sample mean calculation

**Optional**: No

**Module**: sampleMeansCalc.py

**Description**: The non-zero means of each frame’s sample’s replicates is calculated and inserted as a new column for each sample.

**Stage prefix**: ‘13-Meaned\_data’

Step 14 – Mean retention time correction

**Optional**: Yes

**Module**: rtCorrect.py

**Description**: As ‘Stage 11 – Replicate retention time correction’ but for the sample means for each frame.

**Stage prefix**: ‘14-Mean\_Retention\_time\_corrected\_data’

Step 15 – Mass reassignment

**Optional**: No, but user has choice of m/z level or feature level

**Module**: reassignMass.py

**Description**: The m/z within each feature cluster of mass cluster is set to the highest mean of the frame means within the m/z cluster or feature cluster depending on the toggle *featureLevelMassAssignment*.

**Stage prefix**: ‘15-Mass\_reassigned\_data’

Step 16 – Broad retention time contaminant removal

**Optional**: No, but parameters can be amended

**Module**: broadContaminant.py

**Description**: An additional contamination signature is a spread of similar intensity peaks across the retention time range the mass mass cluster. PeakFilter removes these contaminants leaving outlying higher intensity peaks as genuine lipid-like features.

**Stage prefix**: ‘16-Broad\_contaminant\_removed\_data’