XCMS Tutorial

Version 1.0

By Emma Spady, released 2023-05-23

General Notes

When I specify names of files, these should be exactly copied. A capital letter, hidden space, or typo will prevent my script from finding the relevant file. When there's text you need to change, it's best to limit yourself to letters, digits, period, dash and underscore. I will call these the 'friendly characters'.

File Setup

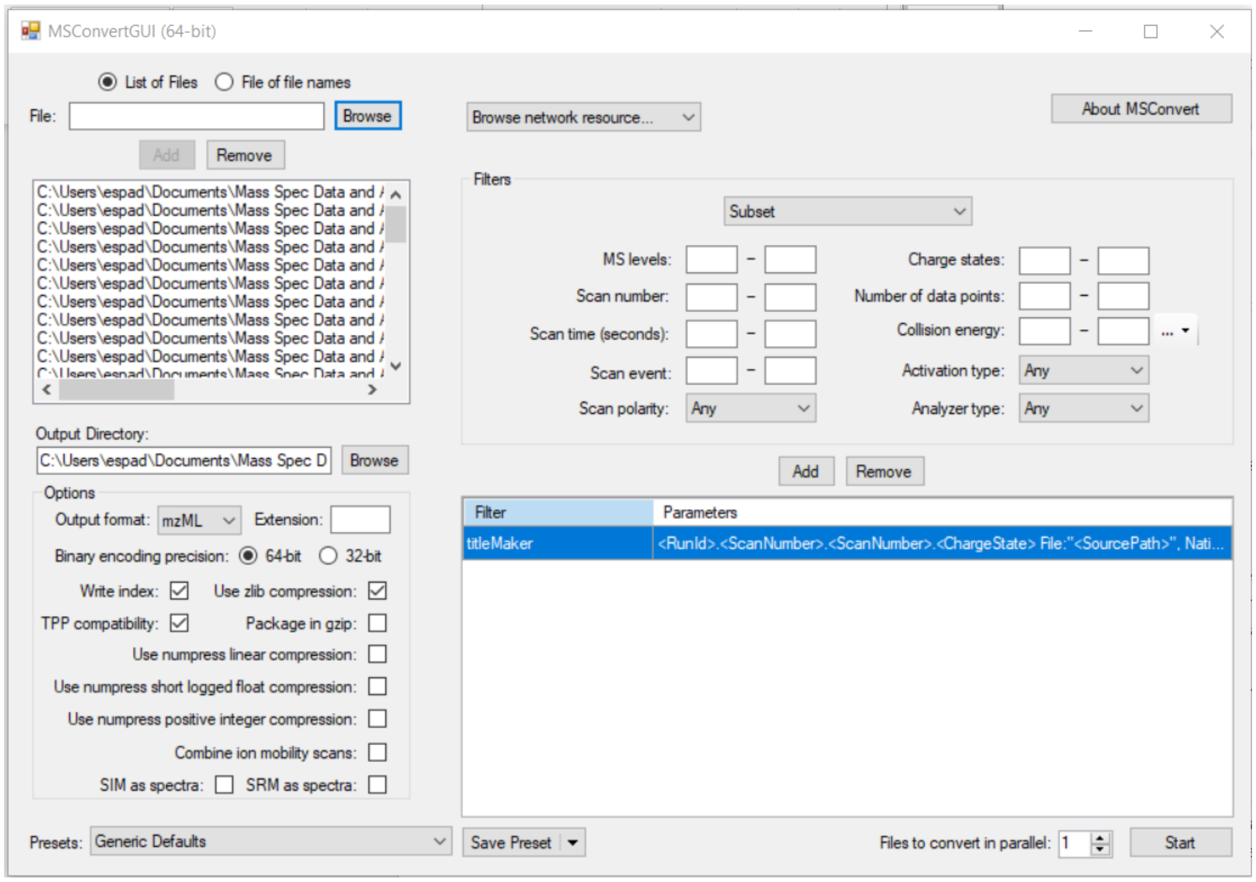
You'll want a fresh directory for this work. Let's call this the 'main directory' or 'maindir' for short. Make a new folder on your local computer, named for your experiment. (I would worry the server connection will slow down processing).

Copy the entire Template1.0 folder, and then rename it for your own work. This includes xcmsMainScript_1.0_template.R, ES_XCMSfunctions_1.0.R, comparekey.csv, conditionkey.csv, pdtable.csv, PwaySet_Espady01.txt, and QCmols_ANP-NEG.csv and "-POS.csv. There should be an empty directory in the maindir called 'rawdata'. Do not rename any files except the main script.

Data file conversion

Decide which data files you want to analyze together. These files should be run on the same instrument, in the same mode, and reasonably close in time, such that retention times can be aligned easily. The column cannot have been replaced between these samples. My current guideline, based on the lab standard retention time drift, is that the samples should be run within 3 weeks of one another. (This is with current low instrument use as of May 2023– this may change as we use our instruments more frequently.) (Eventually I will figure out retention time and possible intensity correction across larger time ranges, but for now, I can only be confident about this scale)

Convert your desired files to .mzml format. I use MSConvertGUI. Browse on the top left to select the Agilent-format ".d" files (they're shown in Windows as folders). Then put the name of your maindir\rawdata folder in the Output Directory box. Here are my other settings, which I'm pretty sure are the default. Press "Start" and wait until all the files have been processed. Then you can close all the windows. You should now see .mzml files in your rawdata directory.



(● List of Files) ○ File of file names

File:

C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and
C:\Users\espad\Documents\Mass Spec Data and

Output Directory: C:\Users\espad\Documents\Mass Spec D

Options

Output format: mzML Extension:

Binary encoding precision: (● 64-bit) ○ 32-bit

Write index: ✓ Use zlib compression: ✓

TPP compatibility: ✓ Package in gzip: ☐

Use numpress linear compression: ☐

Use numpress short logged float compression: ☐

Use numpress positive integer compression: ☐

Combine ion mobility scans: ☐

SIM as spectra: ☐ SRM as spectra: ☐

Presets: Generic Defaults

Subset

MS levels: – Charge states: –

Scan number: – Number of data points: –

Scan time (seconds): – Collision energy: –

Scan event: – Activation type: Any

Scan polarity: Any Analyzer type: Any

Filter	Parameters
titleMaker	<RunId>.<ScanNumber>.<ScanNumber>.<ChargeState> File:"<SourcePath>", Nati...

Files to convert in parallel: 1

Experiment Plan Setup

Now we will set up the three .csv files with experiment information. Remember to keep these file names the same so my script can find them. In addition, be careful what column names you change in these files. In the content, avoid spaces, special characters except - and _, and starting names with digits.

conditionkey.csv

This is where you assign information for each of the sample groups in your experiment. The three essential columns are 'label', 'color', and 'chartgroup'. The first column is a short label for each sample group. Keep this to a few letters long if possible. It will be used to reference samples to one another.

Label requirements: If using QC samples, you should make a sample group called "qc". Then, each label must not match to a substring of another label. To illustrate: don't label one group "treat" and another "untreat". Because my script will search for the charaters 'treat', and it will match to both labels. Instead, I would use 'treat' and 'control'. Similarly, avoid 'drug1' and 'drug10'. (instead, use 'drug01' and 'drug10')

'color' is a hex code for a color for the samples – most important when the script makes chromatograms. 'chartgroup' is a name for the panel grouping for the chromatograms. Each unique 'chartgroup' string will become a separate panel in the plot, with that string as the title. This helps visualize a complicated experiment (preventing what I like to call the 'rainbow spaghetti problem'). I generally group my charts by the drug I used. You'll want to keep each panel to 4 groups or so.

The other columns in conditionkey.csv are for your own use to store treatment information. You can relabel them freely, and add more if you like. I usually put the pH, PZA dose, and RIF dose in these. You might want incubation time, genotype, enzyme added, or other treatments.

pdtable.csv

This is where you match your labels to the file names for each replicate in the sample group. It should have only three columns: ‘sample_name’, ‘sample_group’, and ‘file’.

Let’s start with ‘file’. The files must be in the order that they ran on the instrument. This enables the use of QC samples to help with the retention time adjustment. So I typically use my worklist setup file, made in excel, to help with this part. I paste the column of file names, and then ctrl+f to replace ‘.d’ with ‘.mzml’.

Now add the sample_name. This should match one of the labels that you chose in conditionkey.csv, except with an underscore and a replicate number. If my labels are ‘treat’ and ‘control’, and I have five replicates each, I’d label my files treat_1, treat_2, …, treat_5, control_1, control_2, …, control_5. If you are using QC samples to adjust the retention times, you must name them ‘qc_1’, ‘qc_2’, etc.

Finally, add the sample_group. This is a short descriptive label for the sample group, and should be the same for all of the samples in a group. I usually make a temporary column in my conditionkey.csv file, and use the =CONCAT() function to put together some of the optional columns there – in my case, drug doses and/or pH.

comparekey.csv

This is where you specify which sample groups should be t-tested against one another. Use the same labels as in the conditionkey to specify your groups of choice. Then make up a descriptive name for the comparison in the compareID column. It will become a filename for the volcano plots and such. Finally, put a category for the comparison – it’s helpful to group comparisons together by the ‘category’ column if you have dozens of comparisons like I do. A group might be various RIF doses in the same pH treatment. Use only friendly characters in this column because it will become folder names.

In the demo set, notice how there are some categories with many comparisons, and others with only one comparison each. The individualized categories make it so chromatograms will print for all top 50 hits from just that volcano plot. So it’s good to make individual categories for comparisons you want to explore deeply.

Running the Code

Start by renaming the main script, called xcmsMainScript_1.0_template.R to something meaningful to the experiment. Just for your records, so you know what you did to your data. Then open RStudio, and open the script. If you’re on the shared computer, and it asks, use R version 4.3.0.

General Notes

This script is rather long, and has several customizable segments. Thus, I would suggest running code by highlighting the lines you need and pressing ctrl+enter in Rstudio. You'll then see the selected lines of code run in the terminal below. You can also simply move your cursor to a particular line, then hit ctrl+enter to run it.

There are going to be some slow steps in the analysis if you have a large dataset. If your computer has multiple cores (as most do) you can have the analysis running while you do other things, without slowing it down. This is because multithreaded evaluation isn't available for Windows in xcms, unless it's a computing cluster. Basically, xcms can only use one of your computer's cores at a time anyway, so you can continue using Microsoft Office or web browsing on the other cores. Just don't try to do anything intensive.

After a slow step, you may want to save your progress in case something goes wrong in later steps. For further info, see the heading ####SAVE SYSTEM IMAGE####

####PREP####

Edit the code under these headings, then run this section.

Retitle Code

You can edit information at the top of this code to help you remember which experiment this is set up to analyze. Use the commented lines at the top for your personal notes.

Load libraries

You'll need to pre-install the packages xcms, manhattanly, htmlwidgets, and MatrixGenerics. This only needs to be done once on your computer, using the Rstudio top toolbar under "Tools", "Install Packages".

Set Working Directory

Edit the text here to be the address of your analysis folder, the maindir. You can get it from right-clicking the top of a Windows Explorer window with maindir open, then choosing "Copy Address as Text". Then paste that inside the quotes in the setwd() function, and switch the direction of the slashes to be /. R needs to use /, the forward slash, in the address because the backslash \ is the 'escape character'.

Name Your Experiment

Make a little tag for your experiments using only friendly characters. It will be used to make output files.

Load a Previous Session

If you have done some analysis and you want to load your work, uncomment the line below and run it. Otherwise leave it alone.

####SETTINGS####

You'll now see a series of settings, some of which are commented out. See the notes in the code for which version you should use. Uncomment / comment out the appropriate blocks, then run everything under ####SETTINGS### and the next section. Here's the parts you'll need to check:

Set ppm error and mode

Put the ppm error you want for matching molecules to a formula in this line. Default is 50. Set the mode to "POS" or "NEG" so the formula matching process generates [M+H]+ and [M-H]- masses appropriately.

Retention time range

Choose the ANP or HILIC option.

Peak finding parameters

Choose TOFs or QTOF2

Retention time alignment parameters

Choose Using QC samples or Single Run.

####DATA READ-IN####

Run this section. It loads the data files into R. This make take a couple minutes. It also makes an object my script uses later called the pdextra. It'll have any miscellany that is needed for the next step.

####QUALITY CONTROL PART 1####

This section will help you confirm that you have chosen appropriate peak-finding settings for your data. These are the parameters that I named cwp and mpp. ctrl+F to locate them in the SETTINGS section. You can probably skip this part, but I needed this workflow for myself anyway, so I streamlined it a bit.

First, change `qcmolfile = ""` to specify your set of QC molecules. The ones I have chosen are in QCmols_ANP-NEG.csv or QCmols_ANP-POS.csv. Feel free to edit by adding your own molecules – just make sure you have the same column names and the correct information for your choices. (in particular, a neutral exact mass rather than a m/z or ion mass)

Then choose the sample conditions that you want to check under `checkedconds`. I would suggest checking your negative control and the conditions that are most extreme. That way the molecules I suggest will be most likely to appear in at least one of them. Make a vector using the labels from the condkey. If I want to check conditions H, Z, and A, I write:

```
checkedconds = c("H", "Z", "A")
```

Finally, set the mode to either "NEG" or "POS", and put the ppm error you want on your chromatogram.

Run this section. It will make a set of chromatogram images in the main directory showing your selected QC molecules over the chosen retention time. It may take a little while if you chose >20 conditions.

The chromatogram images are named based on the QC set molecule they show. Each condition appears in its own graph, which have independent intensity axes. This is to help with troubleshooting, since one condition might have an intense peak where another has a small peak. Each sample is in a different color, automatically generated. The squares around the peaks are the identified peaks in the chromatogram.

You want as many of the clear peaks to have an appropriate square around them as you can. You should change the parameters assigned to cwp and mpp until the boxes fit correctly. The code has comments that explain a bit about what each parameter that goes into cwp and mpp controls. If you want to change something, close all images, re-run the SETTINGS section, then re-run the QUALITY CONTROL. This will change the images to be the new peak finding settings. If you want to save the old images, I recommend moving them into a different folder because they get overwritten when you run QUALITY CONTROL.

####XCMS PROCESSES####

Highlight all this section and run it without edits. It may take a few hours, which varies depending on the computer and the analysis settings. See comments in the code itself to see how long it took on my work laptop. While it runs, it should tell you the time and its progress in the console. You'll know Rstudio is working when the > doesn't appear at the console. You'll see a little stop sign in the console window upper right – click it if you need to stop the code.

####QUALITY CONTROL PART 2####

Don't run this section for now. I will expand it in future versions

Eventually, this section will help you check that you have chosen appropriate retention time alignment, peak grouping, and peak fill settings for your data. These are the settings called owp, pdp, and cpap in the code.

To check retention time alignment, run the plot RT alignment.

####UNTARGETED RESULTS####

Run this section. It will make a text file with all the features identified in your samples. They aren't yet filtered by matching to any molecules. It will be called exptkey_AllFeatures.txt.

Then it will make a folder called "AllFeatVolcanoes". The volcano plots will be made in there. Then it will make subfolders for each of your comparekey 'category' groups. These will contain chromatograms with up to 50 of the hits (those features that are over the p value and foldchange thresholds) from each category, generated in order of highest to lowest foldchange. If you want to see them in order, right-click the Windows Explorer window, choose "Sort by", "Date", "Ascending". The volcano plot

The chromatograms take a long time to generate, and I haven't figured out how to make XCMS be a little faster about it. The slow function is featureChromatograms(). I think it's because R is going back to each of the raw data files to pull out the chromatogram. This is the step before they all print out – and it can take a couple hours. I often skip making untargeted chromatograms because they're often not interesting – they might all be fragments / adducts of a drug, for example. So consider skipping all the code under the heading "#Print chromatograms of the top 50 features with the highest foldchange in each test category".

Thankfully, the volcano plots come out first and are fast, so you can look at them while the chromatograms go. You can also explore the list of features significant in any one of your comparisons. It's titled exptkey_SigUnfiltered.txt, and will be in the AllFeatVolcanoes folder.

####FORMULA-TARGETED RESULTS####

Now we will filter the features by their m/z being within 50 ppm of the predicted [M+H]+ or [M-H]- of a long set of chemical formulae. (or whatever error you chose in the settings). I chose these chemical formulae using KEGG pathways, then filtering out molecules in the pathways that didn't have exact mass associated with the formula (basically things with formulae involving R groups). For more info on that script, see my DatabaseViaKeggrest information.

For now, let's use my molecule set. It's called PwaySet_ESpady01.txt. The only columns that this script actually uses are: Formula, ExactMass, Mode, Label, and RT_known. It contains 1,043 unique formulae selected from 46 Mtb pathways. See the Resources for the full list. You can change the PwaySet by loading a different file name. Make sure the formulae are unique – otherwise the code might fart out.

Run this whole section, and it will give you volcano plots and chromatograms organized the same way as the entirely untargeted results. Again, the chromatograms will take some time. The volcano plots will now have a tentative formula and ID assigned to each molecule based on the m/z. It has the KEGG ID as well.

There are ways to use isotope patterns to assign formulae better, but I haven't implemented that yet. In addition, this approach doesn't use RTs. I have written functions that match RTs as well, but I haven't streamlined it yet.

####FULLY-TARGETED RESULTS####

Work in progress. I have been making my scripts work mostly with unique formula sets, so I don't have great ways to handle isomers automatically. Things might be weird if you have repeat formulae in general.

####CUSTOM CHROMATOGRAMS####

For investigating specific features or (m/z, RT) ranges visually. Do not run this code without editing it – the default features and conditions will not match yours.

There are two steps to making a chromatogram in R with xcms – making the chromatogram object, and then making the plot.

Making a Chromatogram Object

XCMSnExp class objects, like xdata_ftg, don't store the information needed to make a full chromatogram. It keeps their size on the disk down, which is helpful for processing speed. Instead, chromatogram-generating functions pull directly from your raw .mzML files. Then they stick that data into a new object – for example, chr_raws in the template code. That new object is of the class XChromatograms or Mchromatograms. Calling chromatogram-generating functions is quite slow, because it opens and closes each sample file. Unless you want to wait for hours, it's important to group as many chromatograms as you can into each function call.

There are two ways I've made chromatogram objects: featureChromatograms() and chromatogram()

featureChromatograms() is for features of interest. It extracts information around your feature groups from all your samples. Even if one particular sample doesn't have a peak in a feature of interest, any intensity in that sample will still be plotted in the chromatogram. To use it, make a vector with feature names, of the format "FT0001", that you want to see, assigned to customfeats. You can put as many feature names as you like, in quotes and separated by commas. Run this line. Then run the lines below, which assigns the chromatogram object to customftchroms. This might take a long time.

chromatogram() extracts m/z and retention time ranges to plot. It's more like a conventional EIC from Profinder. Since it is annoying to manually make tables in the code directly, I've added a template to help you read these in. It's called customEICinfo.csv. Don't change the first row, but you can edit everything else. Unfortunately, I can't print out chromatograms from this yet in a streamlined way. I'm going to need to write a new function that doesn't rely on having feature information.

Plotting the chromatograms

The main way I print out chromatograms is with ChromPrint(). It generates the type of chromatogram I describe in the "Understanding Your Results" section. It's one of my custom functions, putting each feature into its own .png file. Then it separates the samples into panels based on their chartgroup (which you have specified in the conditionkey.csv). It colors each replicate within the sample group based on the conditionkey.csv hex code color. It draws a box of that color around any identified peaks in the feature.

If you know about R's graphics abilities, you might prefer to make your own custom chromatogram plots. The function plot() works on both XChromatograms and MChromatograms classes, and takes most of the standard R graphing arguments.

I'm working on improved ChomsByConds() functionality, but right now it relies on certain prerequisites that make it only suitable for use in Quality Control Part 1.

I'm also working on making a looping chromatogram printer that works with the object that comes out of chromatogram(), because my main one relies on features.

####SAVE SYSTEM IMAGE####

At the end of the file, you'll see a line of code that makes a file saving your R workspace. This is really helpful if you have any slow analysis steps that you don't want to repeat later. I would suggest saving after the xcms processes and whenever chromatograms are generated. It will make a .Rdata file you can then load. When you reopen your session, the line for that is under the ####PREP#### section. You can then pick up where you left off – though the terminal won't have any previous text output, so make sure you take notes so you know where you were.

Understanding Your Results

XCMS terms

Peak: A m/z and RT range that satisfies signal and shape requirements in a single sample. Xcms finds the m/z range first, because that dimension is very sharp (think spikes, not peaks). Then it looks in each m/z slice for RT ranges of interest. Those become your peaks. Defined with findChromPeaks(), and improved with refineChromPeaks(). I haven't really investigated the storage method for individual peak lookup, since the only relevant ones are in Features.

Feature: A set of peaks with similar m/z and RT across many samples. Settings define the acceptable (m/z, RT) errors, and the fraction of samples that must have the peak to bother grouping it. Done with groupChromPeaks(). Each feature gets an internal ID, of the format FT9999. You then use that ID to get information about the feature, like its raw chromatogram or its evaluated intensity in each sample.

Filled Peak: When a feature has been identified, some samples might have a little intensity in relevant m/z,RT range. But it wasn't quite enough, or the right shape, to trigger the peak assignment. So to fix this, xcms can evaluate (either integrating or taking the maximum) of any signal it finds in that m/z,RT coordinate. The m/z is usually very consistent from the other samples, but I think xcms uses the median m/z with the same error that was used in the peak finding step. In the RT dimension, the evaluation zone starts with the median of the lower limit of the RTs from the other samples. The evaluation zone ends with the median of the upper limit of those RTs. Done with fillChromPeaks()

Chromatograms

I set the chromatogram printer such that it shows +1 and -1 minute around the feature of interest. Then each chromset, as defined in your conditionkey, gets its own panel. I automatically scale all panels to the highest peak that is part of the feature. So if you see a trace cut off, that trace isn't part of the feature – could be a warning sign that your peak picking or feature grouping is wrong, or it could just be a nearby more abundant isomer. The colors for each condition are defined in the conditionkey, with all replicates in the same condition being the same color.

The squares around the peaks are the officially identified peaks in the feature. The squares do not cover filled peaks – so you'll get a sense of how well your settings did in picking up the peaks originally. The filled peaks are evaluated in the output data tables, so they're filled in for the volcano plots and any downstream analysis.

Number of Features

The number of features that you get out of this process varies dramatically with the settings. In particular, setting noise/prefilter in CentWaveParam() has a large effect. Putting these too high will make you have fewer peaks, but those peaks will be higher intensity, and thus typically higher quality. (fewer mistakes due to flickery texture on the top, for example). Higher noise setting also makes xcms processes run more quickly, because there are fewer peaks and features to evaluate.

One advantage to having fewer peaks in a volcano analysis is that the multiple hypothesis testing correction becomes less strict. So fewer peaks might mean more are significant. I currently use "BH" (Benjamini-Hochberg). For more info, see https://www.statisticshowto.com/benjamini-hochberg-procedure/ and https://en.wikipedia.org/wiki/False_discovery_rate . PS. If you come up with a reason I should use a different correction, I'd love to hear it. I don't fully understand the difference between the various ranking-threshold-type correction methods.

On the other hand, I'm concerned that PCA or other clustering type algorithms will not work well if you're missing too many peaks. I'm working on how many peaks to expect in a 'good' ANP metabolomics analysis, given a particular dataset. And I'm working on how many peaks you would want for volcanoes vs clustering.

I currently put prefilter at (3, 1.67*noise). So if you want to change noise, remember to change that part of prefilter as well.

Known Issues

Sometimes multiple peaks from a single sample end up in one feature together. I'm working on how that gets analyzed. But I fixed some settings and now it happens less – in particular, bw under PeakDensityParam().

XCMS has some difficulties with broad peaks or groups of peaks very close together. I hand-optimized the peak picking for our ANP chromatography, but it might do better with different settings on HILIC. If you're getting a lot of funny peak shapes in your results, we can discuss. I'll set up a quality control process that helps you check your settings as you go along... at some point. See the vignette at the bottom if you want more details there.

Terms and Resources

Directory: What Windows calls a “folder”. The area where you have your files stored. When you use this script, you’ll need R to access your directory with data and settings files inside it, and save any output to the same directory. So you’ll set your ‘working directory’, a default directory that R operates inside. The relevant function is setwd(). If you want to check the directory R is currently in, call getwd().

Path, Address: synonyms for the location of a directory or file in a computer. Functions that take paths as arguments need the path to be a string. (so you’ll see they’re inside quotes when written out). Windows default paths use the backslash \, but R needs its paths to use the forward slash /. (so Windows will say you’re in C:\Users\Emma\Documents, but R needs that written as “C:/Users/Emma/Documents”. Some useful abbreviations: “.” at the beginning of a path means the current directory you’re in. “..” means the directory above (ie containing) the directory you’re in.

Comment: A comment is a character that causes everything past it in the line of code to not be read by the computer. Commented text appears as light green in RStudio. The comment character for R is #. Comments provide notes on code. They can also allow you to skip or unskip portions of code. To add or remove a # from the beginning of a whole block of text in Rstudio, press ctrl+shift+c. Use this to change the settings to your particular instrument.

Function: They look like ‘functionname(argument)’ in R. The function does a process on the argument. Whenever you see a normal parenthesis in R, that’s a function being used. A common term for running a function is ‘calling’ it.

Class: The type of object associated with some name in R. Could be an integer, a vector, a matrix, a dataframe, a string, a list, a function, etc. A function generally only works on certain classes of objects – imagine the function sqrt(), which takes the square root of its argument, would not work on a string. Because a string is letters, not a number, and you can’t take the square root of letters. Check it for an object named foo by calling class(foo). Class mismatch is a big source of errors.

String: Data that is text in R. Generally denoted by quote marks. The opposite of a number, I guess. Shown as dark green in Rstudio. If a number got read in as a string by accident, you can use as.numeric() to convert it. The function to invert that is as.character(). The class of a string in R is “character”.

Vector, Matrix, Dataframe: R objects that store values in specific indexes. To look at the indexes, you use square brackets and commas. A vector has only one dimension: look for the 5th value in a vector using vector[5]. A matrix or dataframe has two dimensions: look for the 5th row, 3rd column in a matrix using matrix[5,3]. If you want rows 5 through 10 and the third column from that matrix, use a colon like matrix[5:10,3]. If you have a comma with no number in square brackets, you get all the rows or columns. So all columns in the 8th row of a matrix would be matrix[8,]. You can also use row names or column names to find values in these types of objects. For example, dataframe[“Rv3604”, “ratio”] if your dataframe has genes as rownames and types of outputs in the columns. Matrix and vector entries all have to be the same class (like all strings or all numbers). Dataframes can have different classes in their columns, but a given column has to all the be same class.

List: R objects that store any other object in a single dimension index. The objects don't need to be the same class. So you can have a list where the first entry list[[1]] is a matrix, the second entry list[[2]] is a string, and the third entry list[[3]] is a vector. Referenced using double square brackets as shown.

Other classes: There are certain package-specific classes we will use, like the XCMSnExp. That's the object that xcms data comes in when it's being worked on in R.

Package: A downloadable set of R code designed for specific operations. You can install them from the Rstudio toolbar under "Tools". Or using the command line.

Library: The functions inside a package. Once you call 'library(packagename)' you can use functions inside that package.

Resources

Documentation PDF for xcms is here:
https://bioconductor.org/packages/release/bioc/manuals/xcms/man/xcms.pdf

The xcms vignette I based this pipeline on is here:
https://bioconductor.org/packages/release/bioc/vignettes/xcms/inst/doc/xcms.html

All the IDs for the KEGG pathways I got formulae from:

path:mtu00010, path:mtu00020, path:mtu00030, path:mtu00040, path:mtu00051, path:mtu00052, path:mtu00053, path:mtu00061, path:mtu00071, path:mtu00130, path:mtu00220, path:mtu00230, path:mtu00240, path:mtu00250, path:mtu00260, path:mtu00270, path:mtu00280, path:mtu00290, path:mtu00300, path:mtu00310, path:mtu00330, path:mtu00340, path:mtu00350, path:mtu00360, path:mtu00380, path:mtu00400, path:mtu00410, path:mtu00430, path:mtu00470, path:mtu00480, path:mtu00520, path:mtu00620, path:mtu00630, path:mtu00640, path:mtu00650, path:mtu00660, path:mtu00680, path:mtu00730, path:mtu00740, path:mtu00750, path:mtu00760, path:mtu00770, path:mtu00780, path:mtu00785, path:mtu00790, path:mtu00860