

---

# **nPYc Toolbox Documentation**

***Release 1.2.6***

**National Phenome Centre**

**Aug 10, 2021**



<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Introduction to Metabolic Profiling . . . . .	3
1.2	Tutorials . . . . .	3
1.3	Recommended Study Design Elements . . . . .	3
1.4	Datasets . . . . .	4
1.5	Sample Metadata . . . . .	4
1.6	Sample and Feature Masks . . . . .	4
1.7	Reports . . . . .	4
1.8	Batch & Run-Order Correction . . . . .	4
1.9	Multivariate Analysis . . . . .	4
1.10	Normalisation . . . . .	5
1.11	Exporting Data . . . . .	5
1.12	Configuration Files . . . . .	5
1.13	Enumerations . . . . .	5
1.14	Utility Functions . . . . .	5
1.15	Plotting Functions . . . . .	5
<b>2</b>	<b>Metabolic Profiling</b>	<b>7</b>
<b>3</b>	<b>Installation and Tutorials</b>	<b>9</b>
3.1	Installing the nPYc-Toolbox . . . . .	9
3.2	Installing the nPYc-toolbox-tutorials . . . . .	10
3.3	Using the Jupyter Notebooks . . . . .	10
3.4	Tutorial Datasets . . . . .	11
3.5	Preprocessing and Quality Control of LC-MS Data with the nPYc-Toolbox . . . . .	11
3.6	Preprocessing and Quality Control of NMR Data with the nPYc-Toolbox . . . . .	13
3.7	Preprocessing and Quality Control of NMR Targeted Data with the nPYc-Toolbox . . . . .	13
<b>4</b>	<b>Recommended Study Design Elements</b>	<b>15</b>
4.1	Sample and Study Design Nomenclature . . . . .	15
<b>5</b>	<b>Datasets</b>	<b>17</b>
5.1	LC-MS Datasets . . . . .	18
5.2	NMR Datasets . . . . .	19
5.3	Targeted Datasets . . . . .	19
5.4	Dataset Specific Syntax and Parameters . . . . .	20

<b>6</b>	<b>Sample Metadata</b>	<b>45</b>
6.1	CSV Template for Metadata Import . . . . .	45
6.2	Analytical Parameter Extraction . . . . .	47
<b>7</b>	<b>Sample and Feature Masks</b>	<b>49</b>
7.1	Using updateMasks . . . . .	50
7.2	Using excludeSamples and excludeFeatures . . . . .	52
7.3	Using applyMasks and initialiseMasks . . . . .	52
<b>8</b>	<b>Quality Assessment Reports</b>	<b>53</b>
8.1	Sample Summary Report . . . . .	54
8.2	Feature Summary Report: LC-MS Datasets . . . . .	54
8.3	Feature Summary Report: NMR Datasets . . . . .	57
8.4	Feature Summary Report: NMR Targeted Datasets . . . . .	58
8.5	Dataset Specific Reporting Syntax and Parameters . . . . .	60
<b>9</b>	<b>Batch &amp; Run-Order Correction</b>	<b>63</b>
9.1	Batch & Run-Order Correction Assessment . . . . .	64
9.2	Running Batch & Run-Order Correction . . . . .	64
<b>10</b>	<b>Multivariate Analysis</b>	<b>67</b>
10.1	PCA Model . . . . .	67
10.2	Multivariate Analysis Report . . . . .	68
10.3	Interactive Plots . . . . .	72
<b>11</b>	<b>Normalisation</b>	<b>73</b>
11.1	Normalisation Syntax and Parameters . . . . .	73
<b>12</b>	<b>Exporting Data</b>	<b>77</b>
<b>13</b>	<b>Configuration Files</b>	<b>79</b>
13.1	Built-in Configuration SOPs . . . . .	79
13.2	Generation of Targeted SOPs . . . . .	81
<b>14</b>	<b>Enumerations</b>	<b>87</b>
<b>15</b>	<b>Utility Functions</b>	<b>91</b>
<b>16</b>	<b>Plotting Functions</b>	<b>93</b>
<b>17</b>	<b>Plot Gallery</b>	<b>107</b>
<b>18</b>	<b>Glossary</b>	<b>113</b>
	<b>Python Module Index</b>	<b>117</b>
	<b>Index</b>	<b>119</b>

The **nPYc-Toolbox** defines objects for representing, and implements functions to manipulate and display, metabolic profiling datasets.

Contents:



The nPYc-Toolbox is a general Python 3 implementation of the MRC-NIHR National Phenome Centre toolchain for the import, quality-control, and preprocessing of metabolic profiling datasets.

The toolbox is built around creating an object for each imported dataset. This object contains the metabolic profiling data itself, alongside all associated sample and feature metadata; various methods for generating, reporting and plotting important quality control parameters; and methods for pre-processing such as filtering poor quality features or correcting trends in batch and run-order.

The following sections describe these, in approximate order of application, in more detail. However, we strongly recommend downloading and working through the *tutorials* and referring to the documentation when required.

## 1.1 Introduction to Metabolic Profiling

This section provides a brief introduction to metabolic profiling, the analytical background of the technologies used, and the motivation for the implementation of the nPYc-Toolbox.

See *Metabolic Profiling* for details.

## 1.2 Tutorials

This section provides detailed examples of using the nPYc-Toolbox to import, perform quality-control, and preprocess various types of metabolic profiling datasets.

See *Installation and Tutorials* for details.

## 1.3 Recommended Study Design Elements

This section provides an introduction to recommended sample types and analytical study design elements to ensure standardised quality control (QC) procedures and generate high quality datasets.

See *Recommended Study Design Elements* for details.

## 1.4 Datasets

The nPYc-Toolbox is built around a core *Dataset* object, which contains the metabolic profiling data itself, alongside all associated sample and feature metadata; various methods for generating, reporting and plotting important quality control parameters; and methods for pre-processing such as filtering poor quality features or correcting trends in batch and run-order. This section gives details of importing data into a Dataset, and gives details of supported data types.

See *Datasets* for details.

## 1.5 Sample Metadata

Additional study design parameters or sample metadata may be mapped into the Dataset, this section describes the nomenclature and formats for adding data in order to maximise the utility of the toolbox for quality control.

See *Sample Metadata* for details.

## 1.6 Sample and Feature Masks

Each Dataset object contains a sample and feature masks that store whether a sample or feature, respectively, should be used when calculating QC metrics, in the visualisations in the report functions and when exporting the dataset. This section gives details of the masks, the key functions that modify them and how these can be used.

See *Sample and Feature Masks* for details.

## 1.7 Reports

The nPYc-Toolbox offers a series of *reports*, pre-set visualisations comprised of text, figures and tables to describe and summarise the characteristics of the dataset, and help the user assess the overall impact of quality control decisions.

See *Quality Assessment Reports* for details.

## 1.8 Batch & Run-Order Correction

This section describes the tools available to detect, assess and correct longitudinal run-order trends and batch effects in LC-MS datasets.

See *Batch & Run-Order Correction* for details.

## 1.9 Multivariate Analysis

The nPYc-Toolbox provides the capacity to generate a PCA model of the data, and subsequently, to use this to assess data quality, identify potential sample and feature outliers, and determine any potential analytical associations with the main sources of variance in the data.

See *Multivariate Analysis* for details.



## 1.10 Normalisation

This section describes the process for normalising data to correct for dilution effects on global sample intensity.

See *Normalisation* for details.

## 1.11 Exporting Data

This section describes how to export your data (measurements, and feature and sample related metadata).

See *Exporting Data* for details.

## 1.12 Configuration Files

Behaviour of many aspects of the toolbox can be modified in a repeatable manner by creating configuration files, this section describes the default configuration files and their parameters across the different methods, and gives information on how to create your own configuration SOPs.

See *Configuration Files* for details.

## 1.13 Enumerations

The nPYc-Toolbox uses a set of enumerations (complete listings of all possible items in a collection) for common types referenced in profiling experiments.

See *Enumerations* for details.

## 1.14 Utility Functions

This section contains information on the nPYc-Toolbox utility functions, useful functions for working with profiling datasets.

See *Utility Functions* for details.

## 1.15 Plotting Functions

The *Plotting Functions* sections describes the common plots available, both interactive and static version of many plots exist, suitable for use in an interactive setting such as a *Jupyter notebook*, or saving figures for later use.

See the *Plot Gallery* for a visual overview.



---

Metabolic Profiling

---

Metabolic profiling offers a powerful window into the dynamic interaction between an organism's genetic makeup and environmental influences, by assaying the metabolic content of biofluids (Nicholson *et al*<sup>1</sup>). By measuring levels of the products of metabolism, metabolic profiling can capture a real-time picture of an organism's metabolic state, including both genetic factors and non-genetic factors such as environmental, nutritional, and behavioural influences (Holmes *et al*<sup>2</sup>).

The two most common analytic technologies for metabolic profiling are Nuclear Magnetic Resonance (*NMR*) spectroscopy (Dona *et al*<sup>3</sup>), typically directed at the proton spectrum, and hyphenated-mass spectrometry (*MS*), combining a chromatographic separation with mass-spectrometric detection (Lewis *et al*<sup>4</sup>).

NMR spectroscopy provides a highly precise, non-destructive analytical technique, but is hampered by a comparatively low sensitivity. Applied to biofluids for the purpose of profiling, NMR typically yields data as a one-dimensional spectrum, which may be analysed in an untargeted profiling fashion, or further processed to extract lists of quantified compounds from the spectrum and thus treat the data in a targeted manner (Hao *et al*<sup>5</sup>, Ravanbakhsh *et al*<sup>6</sup>).

Mass spectrometry offers a highly sensitive tool for measuring compounds in biofluids, but is limited by the need to resolve compounds according to molecular weight, as many compounds commonly observed in biofluids share

---

<sup>1</sup> Jeremy K Nicholson, John Connelly, John C Lindon and Elaine Holmes. Metabonomics: a platform for studying drug toxicity and gene function. *Nature Reviews Drug Discovery*, 1(2):153-61, 2002. URL: <http://dx.doi.org/10.1038/nrd728>

<sup>2</sup> Elaine Holmes, Ruey Leng Loo, Jeremiah Stamler, Magda Bictash, Ivan KS Yap, Queenie Chan, Timothy MD Ebbels, Maria De Iorio, Ian J Brown, Kirill A Veselkov, Martha L Daviglus, Hugo Kesteloot, Hirotsugu Ueshima, Liancheng Zhao, Jeremy K Nicholson and Paul Elliott. Human metabolic phenotype diversity and its association with diet and blood pressure. *Nature*, 453(7193):396-400, 2008. URL: <http://dx.doi.org/10.1038/nature06882>

<sup>3</sup> Anthony C Dona, Beatriz Jiménez, Hartmut Schäfer, Eberhard Humpfer, Manfred Spraul, Matthew R Lewis, Jake TM Pearce, Elaine Holmes, John C Lindon and Jeremy K Nicholson. Precision High-Throughput Proton NMR Spectroscopy of Human Urine, Serum, and Plasma for Large-Scale Metabolic Phenotyping. *Analytical Chemistry*, 86(19):9887-9894, 2014. URL: <http://dx.doi.org/10.1021/ac5025039>

<sup>4</sup> Matthew R Lewis, Jake TM Pearce, Konstantina Spagou, Martin Green, Anthony C Dona, Ada HY Yuen, Mark David, David J Berry, Katie Chappell, Verena Horneffer-van der Sluis, Rachel Shaw, Simon Lovestone, Paul Elliott, John Shockcor, John C Lindon, Olivier Cloarec, Zoltan Takats, Elaine Holmes and Jeremy K Nicholson. Development and Application of Ultra-Performance Liquid Chromatography-TOF MS for Precision Large Scale Urinary Metabolic Phenotyping. *Analytical Chemistry*, 88(18):9004-9013, 2016. URL: <http://dx.doi.org/10.1021/acs.analchem.6b01481>

<sup>5</sup> Jie Hao, Manuel Liebecke, William Astle, Maria De Iorio, Jacob G Bundy and Timothy MD Ebbels. Bayesian deconvolution and quantification of metabolites in complex 1D NMR spectra using BATMAN. *Nature Protocols*, 9(6):1416–1427, 2014. URL: <http://dx.doi.org/10.1038/nprot.2014.090>

<sup>6</sup> Siamak Ravanbakhsh, Philip Liu, Trent C Bjorndahl, Rupasri Mandal, Jason R Grant, Michael Wilson, Roman Eisner, Igor Sinelnikov, Xiaoyu Hu, Claudio Luchinat, Russell Greiner and David S Wishart. Accurate, Fully-Automated NMR Spectral Profiling for Metabolomics. *PLOS ONE*, 10(5):1-15, 2015. URL: <https://doi.org/10.1371/journal.pone.0124219>

the same chemical formula. This is typically accounted for by coupling MS detection with chromatography (for example, liquid chromatography, LC-MS) in order to further separate molecules by their chromatographic affinity. Such hyphenated methods result in a two dimensional dataset for each sample analysed (mass to charge ratio,  $m/z$  vs. chromatographic affinity, typically as retention time).

Owing to the complexity and volume of LC-MS data, usually a preliminary feature detection step is applied, that reduces the two dimensional raw analytical data to a 1D list of detected features, each of which is characterised by abundance and observed  $m/z$  and retention time. This process may be conducted in both a targeted manner, where the peaks to be integrated are defined in advance, or in an untargeted profiling approach, in which all peak-like features detectable in the data are integrated. There are a wide range of peak-detection algorithms (Spicer *et al*<sup>7</sup>), but all are susceptible to spuriously detecting analytical noise as features, and thus require a stage of de-noising to produce a high final quality dataset.

Both analytical platforms mentioned above are subject to analytical biases and variances in the precision and accuracy of measurements, and this must be accounted for by the inclusion of quality control (QC) measures, in the form of stand alone QC samples, and reference compounds that may be doped into the samples. A well calibrated NMR instrument is expected to have excellent precision, and study specific QC measures are typically limited to the doping of a chemical shift reference and the repeated analysis of a reference sample. Owing to the complex interactions between sample and instrument, however, LC-MS assays, typically show lower measurement precision than NMR, and often exhibit longitudinal signal drifts over the course of an analysis which must be corrected in order to obtain an accurate representation of true levels in each sample.

As alluded to above, regardless of the analytical platforms used to generate measurements, metabolic profiling assays can be broadly separated into two classes: targeted and untargeted profiling analysis. In targeted analyses, the list of compounds to be detected is often defined up-front and the measurements frequently given as absolute quantifications. The data pre-processing strategy is also targeted, as it focuses on the extraction and integration of an expected set of signals. Conversely, in an profiling analysis, the set of compounds measured is expanded to capture as many compounds as possible. Although this approach in theory provides a more complete window into metabolism, the chemical identity of the great majority of detected compounds in the assay will be unknown and, owing to challenges in feature detection, for some features, the measurement precision might be inferior compared to that in a targeted assay.

Targeted and profiling assay each carry their own implied trade-offs in terms of measurement precision, metabolite annotation and quality control strategy. Protocols for the conduct of targeted analyses are well-established (see section ref{targeted}). Quality control in profiling studies has typically been conducted on an *ad-hoc* basis for individual studies, although in recent years there is an increasing push towards the systematisation and automation of pre-processing toolkits (Giacomoni *et al*<sup>8</sup>, Rijswijk *et al*<sup>9</sup>). The nPYc-Toolbox, presented here, is intended to provide a platform for quality control of metabolic profiling datasets, embodying the quality control practices championed by the MRC-NIHR National Phenome Centre, and focusing on the interpretability of the output to both the analysts who generate the data, and the final users who will perform statistical analysis.

---

<sup>7</sup> Rachel Spicer, Reza M Salek, Pablo Moreno, Daniel Cañueto and Christoph Steinbeck. Navigating freely-available software tools for metabolomics analysis. *Metabolomics*, 13(9):106, 2017. URL: <https://doi.org/10.1007/s11306-017-1242-7>

<sup>8</sup> Franck Giacomoni, Gildas Le Corguillé, Mishari Monsoor, Marion Landi, Pierre Pericard, Mélanie Pétéra, Christophe Duperier, Marie Tremblay-Franco, Jean-François Martin, Daniel Jacob, Sophie Goulitquer, Etienne A Thévenot and Christophe Caron. Workflow4Metabolomics: a collaborative research infrastructure for computational metabolomics. *Bioinformatics*, 31(9):1493–1495, 2015. URL: <https://doi.org/10.1093/bioinformatics/btu813>

<sup>9</sup> Merlijn van Rijswijk, Charlie Beirnaert, Christophe Caron, Marta Cascante, Victoria Dominguez, Warwick B Dunn, Timothy MD Ebbels, Franck Giacomoni, Alejandra Gonzalez-Beltran, Thomas Hankemeier, Kenneth Haug, Jose L Izquierdo-Garcia, Rafael C Jimenez, Fabien Jourdan, Namrata Kale, Maria I Klapa, Oliver Kohlbacher, Kairi Koort, Kim Kultima, Gildas Le Corguillé, Pablo Moreno, Nicholas K Moschonas, Steffen Neumann, Claire O'Donovan, Martin Reczko, Philippe Rocca-Serra, Antonio Rosato, Reza M Salek, Susanna-Assunta Sansone, Venkata Satagopam, Daniel Schober, Ruth Shimmo, Rachel A Spicer, Ola Spjuth, Etienne A Thévenot, Mark R Viant, Ralf JM Weber, Egon L Willighagen, Gianluigi Zanetti and Christoph Steinbeck. The future of metabolomics in ELIXIR [version 2; peer review: 3 approved. F1000Research, 6(ELIXIR):1649, 2017. URL: <https://doi.org/10.12688/f1000research.12342.2>

### 3.1 Installing the nPYc-Toolbox

We recommend running the nPYc-Toolbox pipeline using Jupyter notebooks, this can either be done through the data science platform Anaconda (recommended), or alternatively through installing Python and the Jupyter notebooks independently.

#### Using Anaconda - recommended

- Install Anaconda with Python 3.6 or above from [Anaconda Download Link](#)
- Install the nPYc-Toolbox by opening the Anaconda Prompt (see [Getting started with Anaconda](#) for details) and typing 'pip install nPYc', this will install the toolbox alongside any required dependencies and make it available as a general python package
- Note, if you have an older version of Anaconda, it should first be updated by using the Anaconda Prompt and typing 'conda update conda' and 'conda update --all' (it may be necessary to run the prompt as administrator by selecting this option on the right click menu as you open it). For very old versions, it may be necessary to uninstall and reinstall the latest version

#### Using Python and Jupyter

- Install Python 3.6 or above from [Python Download Link](#)
- Install the nPYc-Toolbox by opening a command (Windows) or terminal (macOS) window and typing 'pip install nPYc', this will install the toolbox alongside any required dependencies and make it available as a general python package
- Install Jupyter from [Jupyter Download Link](#)

For advanced users, the toolbox source code can be downloaded directly from the [nPYc-Toolbox GitHub Repository](#)

We strongly recommend additionally downloading the nPYc-toolbox-tutorials, which give detailed worked examples of using the nPYc-Toolbox.

## 3.2 Installing the nPYc-toolbox-tutorials

We strongly recommend downloading the nPYc-toolbox-tutorials, which use Jupyter notebooks to demonstrate the application of the nPYc-Toolbox for the preprocessing and quality control of exemplar LC-MS, NMR and targeted NMR (Bruker IVDr) metabolic profiling datasets. These tutorials work through each step in detail, with links to relevant documentation.

- Download the nPYc-toolbox-tutorials from [nPYc-toolbox-tutorials GitHub repository](#)
- Click on the green ‘Clone or download’ dropdown menu, then the tutorials can be downloaded as a ZIP file and saved in a suitable location

## 3.3 Using the Jupyter Notebooks

### Opening a Jupyter Notebook

Jupyter can be opened from the Anaconda navigator (recommended) or from the command line.

- Using the Anaconda Navigator, launch a Jupyter Notebook session by clicking on the ‘Jupyter Notebook’ icon
- Alternatively, from the command (Windows) or terminal (macOS) window, launch Jupyter by typing ‘jupyter notebook’, for more details see [Running Jupyter](#)
- Either of the options above will result in an instance of the Jupyter Notebooks opening in a browser window

### Running the nPYc-toolbox-tutorials

- Open a Jupyter Notebook session (as described above)
- Click on the ‘File’ tab and navigate to the location where the nPYc-toolbox-tutorials are saved
- Jupyter notebooks save with the file extension ‘ipynb’
- Click on the required Jupyter notebook example (MS, NMR and targeted NMR examples available, as described below) to open in a new browser window
- To run the notebook, click through the cells using the ‘Run’ button
- Notebooks can be reset and restarted using ‘Kernel > Restart & Clear Output’
- Notebooks can be saved using ‘File > Save and Checkpoint’ (notebooks can then be shared, and others will be able to view their contents)
- Full details on using the Jupyter Notebooks can be found here [Jupyter Read the Docs](#)

### Running your own notebook

- Open the required the nPYc-toolbox-tutorials (as described above)
- Select ‘File > Make a copy..’ to make a copy of the notebook
- Select ‘File > Rename..’ to rename the copied notebook
- Replace the file paths in the document with your own data path files
- Run the notebook!

## 3.4 Tutorial Datasets

Exemplar data, as part of the nPYc-toolbox-tutorials, can be downloaded from the [nPYc-toolbox-tutorial GitHub repository](#). This repository contains all the data and files required to run the tutorials (as described below). Data (including all raw LC-MS data files) is also available from the [Metabolights](#) repository under accession number MT-BLS694.

The dataset used in these tutorials (DevSet) is comprised of three distinct pooled samples of human urine, with three additional samples generated by pairwise mixing of each primary sample, resulting in a sample set of six:

- DevSet1
- DevSet2
- DevSet3
- DevSet1v2
- DevSet1v3
- DevSet2v3

The samples were then split into 13 equivalent aliquots, and each independently prepared and measured by ultra-performance liquid chromatography coupled to reversed-phase positive ionisation mode spectrometry (LC-MS, RPOS) and <sup>1</sup>H nuclear magnetic resonance (NMR) spectroscopy according to Phenome Centre protocols (LC-MS: Lewis *et al*<sup>1</sup>, NMR: Dona *et al*<sup>2</sup>). As per these protocols, a pooled QC *Study Reference* sample and independent external reference (*Long-Term Reference*) of a comparable matrix was also acquired to assist in assessing analytical precision.

Urine samples used for generating of the exemplar matrices were collected with informed written consent and ethical approval from the Imperial College Healthcare Tissue Bank (12/WA/0196, project R13053).

## 3.5 Preprocessing and Quality Control of LC-MS Data with the nPYc-Toolbox

This tutorial demonstrates how to use the LC-MS data processing modules of the nPYc-Toolbox, to import and perform some basic preprocessing and quality control of LC-MS data, and to output a final high quality dataset ready for data modeling.

Required files in nPYc-toolbox-tutorials:

- Preprocessing and Quality Control of LC-MS Data with the nPYc-Toolbox.ipynb: Jupyter notebook tutorial for LC-MS RPOS (XCMS) data
- DEVSET U RPOS xcms.csv: feature extracted (XCMS) LC-MS RPOS data (see below)
- DEVSET U RPOS Basic CSV.csv: CSV file containing basic metadata about each of the acquired samples

Additional files (for example, the raw LC-MS data files) can be found in [Metabolights MTBLS694](#)

Feature extraction of the LC-MS dataset (generation of ‘DEVSET U RPOS xcms.csv’ from the raw data files) was conducted with the R package [XCMS](#), using the following script:

<sup>1</sup> Matthew R Lewis, Jake TM Pearce, Konstantina Spagou, Martin Green, Anthony C Dona, Ada HY Yuen, Mark David, David J Berry, Katie Chappell, Verena Horneffer-van der Sluis, Rachel Shaw, Simon Lovestone, Paul Elliott, John Shockcor, John C Lindon, Olivier Cloarec, Zoltan Takats, Elaine Holmes and Jeremy K Nicholson. Development and Application of Ultra-Performance Liquid Chromatography-TOF MS for Precision Large Scale Urinary Metabolic Phenotyping. *Analytical Chemistry*, 88(18):9004-9013, 2016. URL: <http://dx.doi.org/10.1021/acs.analchem.6b01481>

<sup>2</sup> Jake TM Pearce, Toby J Athersuch, Timothy MD Ebbels, John C Lindon, Jeremy K Nicholson and Hector C Keun. Robust Algorithms for Automated Chemical Shift Calibration of 1D <sup>1</sup>H NMR Spectra of Blood Serum. *Analytical Chemistry*, 80(18):7158-62, 2008. URL: <http://dx.doi.org/10.1021/ac8011494>

```
##
# NPC Reverse-Phase Urine XCMS params
##

#####
###---SAMPLESET-DEPENDENT VARIABLES---###
#####

dataDirectory <- "/Volumes/Promise R6/Raw_Data/mzMLRPOS/"
savePath <- "/Volumes/Promise R6/Raw_Data/Example U RPOS XCMS.csv"

workers <- 8

setwd(dataDirectory)

#####
###-----DATA EXTRACTION-----###
#####

library(xcms)

### centWave peak detection: suitable algorithm for high resolution LC/ToF data in
↳centroid mode.
### note the parameters below have been optimised for Xevo G2-S data originating from
↳the NPC Urine RP analysis in POS mode

ds <- xcmsSet(method="centWave",
  noise=600,
  ppm=25,
  prefilter=c(8, 6000),
  snthresh = 10,
  peakwidth=c(1.5,5),
  mzdiff=0.01,
  mzCenterFun="wMean",
  integrate=2,
  lock=F,
  fitgauss=F,
  BPPARAM=SnowParam (workers = workers), # number of core processors
)

# Matches ("groups") peaks across samples (rtCheck = maximum amount of time from the
↳median RT)

# density method
gds<-group(ds, method="density",
  minfrac=0,
  minsamp=0,
  bw=1,
  mzwid=0.01,
  sleep=0
)

# identify peak groups and integrate samples
fds <- fillPeaks(gds, method="chrom", BPPARAM=SnowParam (workers = workers))

write.csv(peakTable(fds), file=savePath)
```



## 3.6 Preprocessing and Quality Control of NMR Data with the nPYc-Toolbox

This tutorial demonstrates how to use the NMR data processing modules of the nPYc-Toolbox, to import and perform some basic preprocessing and quality control of NMR data, and to output a final high quality dataset ready for data modeling.

Required files in nPYc-toolbox-tutorials:

- Preprocessing and Quality Control of NMR Data with the nPYc-Toolbox.ipynb: Jupyter notebook tutorial for NMR (Bruker) data
- DEVSET U 1D NMR raw data files: folder containing the 1D NMR raw data files (Bruker format)
- DEVSET U 1D NMR Basic CSV.csv: CSV file containing basic metadata about each of the acquired samples

## 3.7 Preprocessing and Quality Control of NMR Targeted Data with the nPYc-Toolbox

This tutorial demonstrates how to use the NMR targeted data processing modules of the nPYc-Toolbox to import and perform some basic quality control of outputs from the Bruker IVDr targeted quantification methods and generate a final high quality dataset ready for data modeling.

Required files in nPYc-toolbox-tutorials:

- Preprocessing and Quality Control of Targeted NMR Data (Bruker IVDr) with the nPYc-toolbox.ipynb: Jupyter notebook tutorial for targeted NMR (Bruker IVDr) data
- DEVSET U 1D NMR raw data files: folder containing the 1D NMR raw data files and the Bruker IVDr xml quantification files
- DEVSET U 1D NMR IVDr Basic CSV.csv: CSV file containing basic metadata about each of the acquired samples



---

## Recommended Study Design Elements

---

For the purpose of standardising quality control (QC) procedures within the pipeline and generating high quality datasets, the nPYc-Toolbox defines a recommended set of reference sample types and design elements, based on quality control criteria previously described (Dona *et al*<sup>1</sup>, Lewis *et al*<sup>2</sup>).

One key element in this design is the use of a pooled QC sample, comprised of a mixture of aliquots taken from every sample in the study. The nature of the pooled sample, as a physical average of all samples in the study, guarantees that it will contain representative levels of the majority of compounds present in the samples, including previously unobserved molecules, which is particularly important in profiling studies where the constituents of the sample matrix are not known up-front.

This comprehensiveness allows the pooled QC to be useful in many ways, including, for example, in the generation of measures of analytical precision, such as calculating *Relative Standard Deviation*, accuracy, for example, in calculating *Correlation to Dilution* and to detect and potentially remove analytical batch and run-order effects (see *Batch & Run-Order Correction*).

The following section describes recommended study design elements and key reference QC samples in more detail, alongside how these samples are defined when using the nPYc-Toolbox.

### 4.1 Sample and Study Design Nomenclature

The nPYc toolbox uses the following nomenclature when defining sample types and analytical study design elements. Certain terms are defined and controlled in the *enumerations* module.

Fig. 1: Generation of samples

---

<sup>1</sup> Anthony C Dona, Beatriz Jiménez, Hartmut Schäfer, Eberhard Humpfer, Manfred Spraul, Matthew R Lewis, Jake TM Pearce, Elaine Holmes, John C Lindon and Jeremy K Nicholson. Precision High-Throughput Proton NMR Spectroscopy of Human Urine, Serum, and Plasma for Large-Scale Metabolic Phenotyping. *Analytical Chemistry*, 86(19):9887-9894, 2014. URL: <http://dx.doi.org/10.1021/ac5025039>

<sup>2</sup> Matthew R Lewis, Jake TM Pearce, Konstantina Spagou, Martin Green, Anthony C Dona, Ada HY Yuen, Mark David, David J Berry, Katie Chappell, Verena Horneffer-van der Sluis, Rachel Shaw, Simon Lovestone, Paul Elliott, John Shockcor, John C Lindon, Olivier Cloarec, Zoltan Takats, Elaine Holmes and Jeremy K Nicholson. Development and Application of Ultra-Performance Liquid Chromatography-TOF MS for Precision Large Scale Urinary Metabolic Phenotyping. *Analytical Chemistry*, 88(18):9004-9013, 2016. URL: <http://dx.doi.org/10.1021/acs.analchem.6b01481>

The hierarchy of sample generation, *Study samples* are generated from *participants* at one or more *sampling events*. These sample are then *assayed* by one or more methods, generating a unique dataset for each *sample assay*.

In order to estimate analytical quality in a robust and extensible fashion, the nPYc-Toolbox characterises the samples constituting a study by two parameters; the sample type, i.e., the source and composition of the sample, and the assay role, the rationale for a specific acquisition of data.

Sample Types are described in detail here *SampleType*, the most common are:

- ‘Study Sample’ comprise the study in question
- ‘Study Pool’ a mixture made from pooling aliquots from all/some study samples
- ‘External Reference’ a sample of a comparable matrix to the study samples, but not derived from samples acquired as part of the study

Assay Roles are described in detail here *AssayRole*, the most common are:

- ‘Assay’ form the core of an analysis
- ‘Precision Reference’ acquired to characterise analytical variability
- ‘Linearity Reference’ samples used assess the linearity of response (or *Correlation to Dilution*) in the dataset

The main samples comprising the study are named *Study Sample* (SS), and are a *Study Sample*, *Assay* combination.

In addition, common combinations of *Sample Type* and *Assay Role* are defined within the nPYc-Toolbox and used to characterise data quality, these include:

- *Study Reference* (SR): A *Study Pool*, *Precision Reference* combination used to assess analytical stability across the acquisition run (such as *Relative Standard Deviation*)
- *Long-Term Reference* (LTR): An *External Reference*, *Precision Reference* combination used to assess analytical stability across the acquisition run, and furthermore between different studies
- *Serial Dilution Sample* (SRD): A *Study Pool*, *Linearity Reference* combination used to assess linearity of response, often by repeated injection at varying concentrations or levels of dilution (see *Correlation to Dilution*)

When using the nPYc-Toolbox, acquired samples can be matched to their experimental details (for example, reference sample type or associated biological metadata) as described in the *Sample Metadata* section.

---

## Datasets

---

The nPYc-Toolbox is built around creating an object for each imported dataset. This object contains the metabolic profiling data itself, alongside all associated sample and feature metadata; various methods for generating, reporting and plotting important quality control parameters; and methods for pre-processing such as filtering poor quality features or correcting trends in batch and run-order.

The first step in creating an nPYc-Toolbox object is to import the acquired data, creating a *Dataset* specific for the data type:

- *MSDataset* for LC-MS profiling data
- *NMRDataset* for NMR profiling data
- *TargetedDataset* for targeted datasets

For example, to import LC-MS data into a *MSDataset* object:

```
msData = nPYc.MSDataset('path to data')
```

Depending on the data type, the Dataset can be set up directly from the raw data, from common interchange formats, or from the outputs of popular data-processing tools. The supported data types are described in more detail in the data specific sections below.

When importing the data, default parameters, for example, specific parameters such as the number of points to interpolate NMR data into, or more generally the format to save figures as, are loaded from the *Configuration Files*. These parameters are subsequently saved in the *Attributes* dictionary and used throughout subsequent implementation of the pipeline.

For example, for NMR data, the nPYc-Toolbox contains two default configuration files, 'GenericNMRUrine' and 'GenericNMRBlood' for urine and blood datasets respectively, therefore, to import NMR spectra from urine samples the *sop* parameter would be:

```
nmrData = nPYc.NMRDataset('path to data', sop='GenericNMRurine')
```

A full list of the parameters for each dataset type is given in the *Built-in Configuration SOPs*. If different values are required, these can be modified directly in the appropriate *SOP* file, or alternatively they can be set by the user by modifying the required 'Attribute', either at import, or by subsequent direct modification in the pipeline. For example,

to set the line width threshold (*LWFailThreshold*) to subsequently flag NMR spectra with line widths not meeting this value:

```
# EITHER, set the required value (here 0.8) at import
nmrData = nPYc.NMRDataset(rawDataPath, pulseProgram='noesyggpprd', LWFailThreshold=0.
↪8)

# OR, set the *Attribute* directly (after importing nmrData)
nmrData.Attributes['LWFailThreshold'] = 0.8
```

Dataset objects have several key attributes, including:

- *sampleMetadata*: A  $n \times p$  pandas dataframe of sample identifiers and sample associated metadata (each row here corresponds to a row in the *intensityData* file)
- *featureMetadata*: A  $m \times q$  pandas dataframe of feature identifiers and feature associated metadata (each row here corresponds to a column in the *intensityData* file)
- *intensityData*: A  $n \times m$  numpy matrix of measurements, where each row and column respectively correspond to a the measured intensity of a specific sample feature
- *sampleMask*: A  $n$  numpy boolean vector where *True* and *False* flag samples for inclusion or exclusion respectively
- *featureMask*: A  $m$  numpy boolean vector where *True* and *False* flag features for inclusion or exclusion respectively

Fig. 1: Structure of the key attributes of a *Dataset* object. Of note, rows in the *featureMetadata* Dataframe correspond to columns in the *intensityData* matrix.

Once created, you can query the number of features or samples it contains by running:

```
dataset.noFeatures
dataset.noSamples
```

Or directly inspect the sample or feature metadata, and the raw measurements:

```
dataset.sampleMetadata
dataset.featureMetadata
dataset.intensityData
```

For more details on using the sample and feature masks see [Sample and Feature Masks](#).

It is possible to add additional study design parameters or sample metadata into the Dataset using the *addSampleInfo()* method (see [Sample Metadata](#) for details).

For full method specific details see [Installation and Tutorials](#).

## 5.1 LC-MS Datasets

The toolbox is designed to be agnostic to the source of peak-picked profiling datasets, currently supporting the outputs of *XCMS* (Tautenhahn *et al*<sup>1</sup>), *Bruker Metaboscope*, and *Progenesis QI*, but simply expandable to data from other

<sup>1</sup> Ralf Tautenhahn, Christoph Bottcher and Steffen Neumann. Highly sensitive feature detection for high resolution LC/MS. BMC Bioinformatics, 9:504, 2008. URL: <https://doi.org/10.1186/1471-2105-9-504>

peak-pickers. Current best-practices in quality control of profiling LC-MS (Want *et al*<sup>2</sup>, Dunn *et al*<sup>3</sup>, Lewis *et al*<sup>4</sup>) data are applied, including utilising repeated injections of *Study Reference* samples in order to calculate analytical precision for the measurement of each feature (*Relative Standard Deviation*), and a serial dilution of the reference sample to assess the linearity of response (*Correlation to Dilution*), for full details see *Feature Summary Report: LC-MS Datasets*.

Study Reference samples are also used (in conjunction with *Long-Term Reference* samples if available) to assess and correct trends in batch and run-order (*Batch & Run-Order Correction*). Additionally, both RSD and correlation to dilution are used to filter features to retain only those measured with a high precision and accuracy (*Sample and Feature Masks*).

## 5.2 NMR Datasets

The nPYc-Toolbox supports input of processed Bruker GmbH format 1D experiments. Upon import, each spectrum's chemical shift axis is calibrated to a reference peak (Pearce *et al*<sup>5</sup>), and all spectra interpolated onto a common scale, with full parameters as per the *NMRDataset Objects configuration SOPs*. The toolbox supports automated calculation of the quality control metrics described previously (Dona *et al*<sup>6</sup>), including assessments of line-width, water suppression quality, and baseline stability, for full details see *Feature Summary Report: NMR Datasets*.

## 5.3 Targeted Datasets

The TargetedDataset represents quantitative datasets where compounds are already identified, the exactitude of the quantification can be established, units are known and calibration curve or internal standards are employed (Lee *et al*<sup>7</sup>). It implements a set of reports and data consistency checks to assist analysts in assessing the presence of batch effects, applying limits of quantification (LOQ), standardizing the linearity range over multiple batches, and determining and visualising the accuracy and precision of each measurement, for more details see *Feature Summary Report: NMR Targeted Datasets*.

The nPYc-Toolbox supports input of both MS-derived targeted datasets (tutorial and further documentation in progress), and two Bruker proprietary human biofluid quantification platforms (IVDr algorithms) that generate targeted outputs from the NMR profiling data, *BI-LISA* for quantification of Lipoproteins (blood samples only) and *BIQUANT-PS* and *BIQUANT-UR* for small molecule metabolites (for blood and urine respectively).

<sup>2</sup> Elizabeth J Want, Ian D Wilson, Helen Gika, Georgios Theodoridis, Robert S Plumb, John Shockcor, Elaine Holmes and Jeremy K Nicholson. Global metabolic profiling procedures for urine using UPLC-MS. *Nature Protocols*, 5(6):1005-18, 2010. URL: <http://dx.doi.org/10.1038/nprot.2010.50>

<sup>3</sup> Warwick B Dunn, David Broadhurst, Paul Begley, Eva Zelena, Sue Francis-McIntyre, Nadine Anderson, Marie Brown, Joshau D Knowles, Antony Halsall, John N Haselden, Andrew W Nicholls, Ian D Wilson, Douglas B Kell, Royston Goodacre and The Human Serum Metabolome (HUSERMET) Consortium. Procedures for large-scale metabolic profiling of serum and plasma using gas chromatography and liquid chromatography coupled to mass spectrometry. *Nature Protocols*, 6(7):1060-83, 2011. URL: <http://dx.doi.org/10.1038/nprot.2011.335>

<sup>4</sup> Matthew R Lewis, Jake TM Pearce, Konstantina Spagou, Martin Green, Anthony C Dona, Ada HY Yuen, Mark David, David J Berry, Katie Chappell, Verena Horneffer-van der Sluis, Rachel Shaw, Simon Lovestone, Paul Elliott, John Shockcor, John C Lindon, Olivier Cloarec, Zoltan Takats, Elaine Holmes and Jeremy K Nicholson. Development and Application of Ultra-Performance Liquid Chromatography-TOF MS for Precision Large Scale Urinary Metabolic Phenotyping. *Analytical Chemistry*, 88(18):9004-9013, 2016. URL: <http://dx.doi.org/10.1021/acs.analchem.6b01481>

<sup>5</sup> Jake TM Pearce, Toby J Athersuch, Timothy MD Ebbels, John C Lindon, Jeremy K Nicholson and Hector C Keun. Robust Algorithms for Automated Chemical Shift Calibration of 1D 1H NMR Spectra of Blood Serum. *Analytical Chemistry*, 80(18):7158-62, 2008. URL: <http://dx.doi.org/10.1021/ac8011494>

<sup>6</sup> Anthony C Dona, Beatriz Jiménez, Hartmut Schäfer, Eberhard Humpfer, Manfred Spraul, Matthew R Lewis, Jake TM Pearce, Elaine Holmes, John C Lindon and Jeremy K Nicholson. Precision High-Throughput Proton NMR Spectroscopy of Human Urine, Serum, and Plasma for Large-Scale Metabolic Phenotyping. *Analytical Chemistry*, 86(19):9887-9894, 2014. URL: <http://dx.doi.org/10.1021/ac5025039>

<sup>7</sup> Jean W Lee, Viswanath Devanarayan, Yu Chen Barrett, Russell Weiner, John Allinson, Scott Fountain, Stephen Keller, Ira Weinryb, Marie Green, Larry Duan, James A Rogers, Robert Millham, Peter J O'Brien, Jeff Sailstad, Masood Khan, Chad Ray and John A Wagner. Fit-for-purpose method development and validation for successful biomarker measurement. *Pharmaceutical Research*, 23(2):312-28, 2006. URL: <http://dx.doi.org/10.1007/s11095-005-9045-3>

## 5.4 Dataset Specific Syntax and Parameters

The main function parameters (which may be of interest to advanced users) are as follows:

Note, the Dataset object serves as a common parent to *MSDataset*, *TargetedDataset*, and *NMRDataset*, and should not typically be instantiated independently.

```
class nPYc.objects.Dataset (sop='Generic', sopPath=None, **kwargs)
```

Base class for nPYc dataset objects.

### Parameters

- **sop** (*str*) – Load configuration parameters from specified SOP JSON file
- **sopPath** – By default SOPs are loaded from the `nPYc/StudyDesigns/SOP/` directory, if not `None` the directory specified in `sopPath=` will be searched before the builtin SOP directory.

**featureMetadata = None**

$m \times q$  pandas dataframe of feature identifiers and metadata

The featureMetadata table can include any datatype that can be placed in a pandas cell, However the toolbox assumes certain prerequisites on the following columns in order to function:

Col- umn	dtype	Usage
Fea- ture Name	str or float	ID of the <i>feature</i> measured in this column. Each 'Feature Name' must be unique in the table. If 'Feature Name' is numeric, the columns should be sorted in ascending or descending order.

**sampleMetadata = None**

$n \times p$  dataframe of sample identifiers and metadata.

The sampleMetadata table can include any datatype that can be placed in a pandas cell, However the toolbox assumes certain prerequisites on the following columns in order to function:



Column	dtype	Usage
Sample ID	str	ID of the <i>sampling event</i> generating this sample
Assay-Role	<i>AssayRole</i>	Defines the role of this assay
Sample-Type	<i>SampleType</i>	Defines the type of sample acquired
Sample File Name	str	<i>Unique file name</i> for the analytical data
Sample Base Name	str	<i>Common identifier</i> that links analytical data to the <i>Sample ID</i>
Dilution	float	Where <i>AssayRole</i> is <i>LinearityReference</i> , the expected abundance is indicated here
Batch	int	Acquisition batch
Correction Batch	int	When detecting and correcting for <i>batch</i> and <i>Run-Order</i> effects, run-order effects are characterised within samples sharing the same <i>Correction Batch</i> , while batch effects are detected between distinct values
Acquired Time	datetime.datetime	Date and time of acquisition of raw data
Run order	int	Order of sample acquisition
Exclusion Details	str	Details of reasoning if marked for exclusion
Meta-data Available	bool	Records which samples had metadata provided with the <code>.addSampleInfo()</code> method

**featureMask = None**

*m* element vector, with `True` representing features to be included in analysis, and `False` those to be excluded

**sampleMask = None**

*p* element vector, with `True` representing samples to be included in analysis, and `False` those to be excluded

**AnalyticalPlatform = None**

*VariableType* enum specifying the type of data represented.

**Attributes = None**

Dictionary of object configuration attributes, including those loaded from *SOP files*.

Defined attributes are as follows:

Key	dtype	Usage
'dpi'	positive int	Raster resolution when plotting figures
'figureSize'	positive (float, float)	Size to plot figures
'figureFormat'	str	Format to save figures in
'histBins'	positive int	Number of bins to use when drawing histograms
'Feature Names'	Column <i>featureMetadata</i> in	ID of the primary feature name

**intensityData**

$n \times m$  numpy matrix of measurements

**noSamples**

**Returns** Number of samples in the dataset ( $n$ )

**Return type** int

**noFeatures**

**Returns** Number of features in the dataset ( $m$ )

**Return type** int

**log**

Return log entries as a string.

**name**

Returns or sets the name of the dataset. *name* must be a string

**Normalisation**

Normaliser object that transforms the measurements in *intensityData*.

**validateObject** (*verbose=True, raiseError=False, raiseWarning=True*)

Checks that all the attributes specified in the class definition are present and of the required class and/or values. Checks for attributes existence and type. Check for mandatory columns existence, but does not check the column values (type or uniqueness). If 'sampleMetadataExcluded', 'intensityDataExcluded', 'featureMetadataExcluded' or 'excludedFlag' exist, the existence and number of exclusions (based on 'sampleMetadataExcluded') is checked

**Parameters**

- **verbose** (*bool*) – if True the result of each check is printed (default True)
- **raiseError** (*bool*) – if True an error is raised when a check fails and the validation is interrupted (default False)
- **raiseWarning** (*bool*) – if True a warning is raised when a check fails

**Returns** True if the Object conforms to basic *Dataset*

**Return type** bool

**Raises**

- **TypeError** – if the Object class is wrong
- **AttributeError** – if self.Attributes does not exist
- **TypeError** – if self.Attributes is not a dict
- **AttributeError** – if self.Attributes['Log'] does not exist
- **TypeError** – if self.Attributes['Log'] is not a list

- **AttributeError** – if self.Attributes['dpi'] does not exist
- **TypeError** – if self.Attributes['dpi'] is not an int
- **AttributeError** – if self.Attributes['figureSize'] does not exist
- **TypeError** – if self.Attributes['figureSize'] is not a list
- **ValueError** – if self.Attributes['figureSize'] is not of length 2
- **TypeError** – if self.Attributes['figureSize'][0] is not a int or float
- **TypeError** – if self.Attributes['figureSize'][1] is not a int or float
- **AttributeError** – if self.Attributes['figureFormat'] does not exist
- **TypeError** – if self.Attributes['figureFormat'] is not a str
- **AttributeError** – if self.Attributes['histBins'] does not exist
- **TypeError** – if self.Attributes['histBins'] is not an int
- **AttributeError** – if self.Attributes['noFiles'] does not exist
- **TypeError** – if self.Attributes['noFiles'] is not an int
- **AttributeError** – if self.Attributes['quantiles'] does not exist
- **TypeError** – if self.Attributes['quantiles'] is not a list
- **ValueError** – if self.Attributes['quantiles'] is not of length 2
- **TypeError** – if self.Attributes['quantiles'][0] is not a int or float
- **TypeError** – if self.Attributes['quantiles'][1] is not a int or float
- **AttributeError** – if self.Attributes['sampleMetadataNotExported'] does not exist
- **TypeError** – if self.Attributes['sampleMetadataNotExported'] is not a list
- **AttributeError** – if self.Attributes['featureMetadataNotExported'] does not exist
- **TypeError** – if self.Attributes['featureMetadataNotExported'] is not a list
- **AttributeError** – if self.Attributes['analyticalMeasurements'] does not exist
- **TypeError** – if self.Attributes['analyticalMeasurements'] is not a dict
- **AttributeError** – if self.Attributes['excludeFromPlotting'] does not exist
- **TypeError** – if self.Attributes['excludeFromPlotting'] is not a list
- **AttributeError** – if self.VariableType does not exist
- **AttributeError** – if self.\_Normalisation does not exist
- **TypeError** – if self.\_Normalisation is not the Normaliser ABC
- **AttributeError** – if self.\_name does not exist
- **TypeError** – if self.\_name is not a str
- **AttributeError** – if self.\_intensityData does not exist
- **TypeError** – if self.\_intensityData is not a numpy.ndarray
- **AttributeError** – if self.sampleMetadata does not exist
- **TypeError** – if self.sampleMetadata is not a pandas.DataFrame
- **LookupError** – if self.sampleMetadata does not have a Sample File Name column

- **LookupError** – if self.sampleMetadata does not have an AssayRole column
- **LookupError** – if self.sampleMetadata does not have a SampleType column
- **LookupError** – if self.sampleMetadata does not have a Dilution column
- **LookupError** – if self.sampleMetadata does not have a Batch column
- **LookupError** – if self.sampleMetadata does not have a Correction Batch column
- **LookupError** – if self.sampleMetadata does not have a Run Order column
- **LookupError** – if self.sampleMetadata does not have a Sample ID column
- **LookupError** – if self.sampleMetadata does not have a Sample Base Name column
- **LookupError** – if self.sampleMetadata does not have an Acquired Time column
- **LookupError** – if self.sampleMetadata does not have an Exclusion Details column
- **AttributeError** – if self.featureMetadata does not exist
- **TypeError** – if self.featureMetadata is not a pandas.DataFrame
- **LookupError** – if self.featureMetadata does not have a Feature Name column
- **AttributeError** – if self.sampleMask does not exist
- **TypeError** – if self.sampleMask is not a numpy.ndarray
- **ValueError** – if self.sampleMask are not bool
- **AttributeError** – if self.featureMask does not exist
- **TypeError** – if self.featureMask is not a numpy.ndarray
- **ValueError** – if self.featureMask are not bool
- **AttributeError** – if self.sampleMetadataExcluded does not exist
- **TypeError** – if self.sampleMetadataExcluded is not a list
- **AttributeError** – if self.intensityDataExcluded does not exist
- **TypeError** – if self.intensityDataExcluded is not a list
- **ValueError** – if self.intensityDataExcluded does not have the same number of exclusions as self.sampleMetadataExcluded
- **AttributeError** – if self.featureMetadataExcluded does not exist
- **TypeError** – if self.featureMetadataExcluded is not a list
- **ValueError** – if self.featureMetadataExcluded does not have the same number of exclusions as self.sampleMetadataExcluded
- **AttributeError** – if self.excludedFlag does not exist
- **TypeError** – if self.excludedFlag is not a list
- **ValueError** – if self.excludedFlag does not have the same number of exclusions as self.sampleMetadataExcluded

**initialiseMasks ()**

Re-initialise *featureMask* and *sampleMask* to match the current dimensions of *intensityData*, and include all samples.

**updateMasks** (*filterSamples=True, filterFeatures=True, sampleTypes=[<SampleType.StudySample>, <SampleType.StudyPool>, <SampleType.ExternalReference>, <SampleType.MethodReference>, <SampleType.ProceduralBlank>], assayRoles=[<AssayRole.Assay>, <AssayRole.PrecisionReference>, <AssayRole.LinearityReference>, <AssayRole.Blank>], \*\*kwargs)*

Update *sampleMask* and *featureMask* according to parameters.

*updateMasks()* sets *sampleMask* or *featureMask* to False for those items failing analytical criteria.

---

**Note:** To avoid reintroducing items manually excluded, this method only ever sets items to False, therefore if you wish to move from more stringent criteria to a less stringent set, you will need to reset the mask to all True using *initialiseMasks()*.

---

#### Parameters

- **filterSamples** (*bool*) – If False don't modify *sampleMask*
- **filterFeatures** (*bool*) – If False don't modify *featureMask*
- **sampleTypes** (*SampleType*) – List of types of samples to retain
- **sampleRoles** (*AssayRole*) – List of assays roles to retain

#### **applyMasks()**

Permanently delete elements masked (those set to False) in *sampleMask* and *featureMask*, from *featureMetadata*, *sampleMetadata*, and *intensityData*.

#### **addSampleInfo** (*descriptionFormat=None, filePath=None, filetype=None, \*\*kwargs*)

Load additional metadata and map it in to the *sampleMetadata* table.

Possible options:

- **'Basic CSV'** Joins the *sampleMetadata* table with the data in the *csv* file at *filePath=*, matching on the 'Sample File Name' column in both (see *Sample Metadata*).
- **'Filenames'** Parses sample information out of the filenames, based on the named capture groups in the regex passed in *filenamespec*
- **'Raw Data'** Extract analytical parameters from raw data files
- **'ISATAB'** ISATAB study designs

#### Parameters

- **descriptionFormat** (*str*) – Format of metadata to be added
- **filePath** (*str*) – Path to the additional data to be added

Raises **NotImplementedError** – if the *descriptionFormat* is not understood

#### **addFeatureInfo** (*filePath=None, descriptionFormat=None, featureId=None, \*\*kwargs*)

Load additional metadata and map it in to the *featureMetadata* table.

Possible options:

- **'Reference Ranges'** JSON file specifying upper and lower reference ranges for a feature.

#### Parameters

- **filePath** (*str*) – Path to the additional data to be added

- **descriptionFormat** (*str*) –
- **featureId** (*str*) – Unique feature Id field in the metadata file provided to match with internal Feature Name

Raises **NotImplementedError** – if the descriptionFormat is not understood

**excludeSamples** (*sampleList*, *on*='Sample File Name', *message*='User Excluded')

Sets the *sampleMask* for the samples listed in *sampleList* to `False` to mask them from the dataset.

#### Parameters

- **sampleList** (*list*) – A list of sample IDs to be excluded
- **on** (*str*) – name of the column in *sampleMetadata* to match *sampleList* against, defaults to 'Sample File Name'
- **message** (*str*) – append this message to the 'Exclusion Details' field for each sample excluded, defaults to 'User Excluded'

**Returns** a list of IDs passed in *sampleList* that could not be matched against the sample IDs present

**Return type** list

**excludeFeatures** (*featureList*, *on*='Feature Name', *message*='User Excluded')

Masks the features listed in *featureList* from the dataset.

#### Parameters

- **featureList** (*list*) – A list of feature IDs to be excluded
- **on** (*str*) – name of the column in *featureMetadata* to match *featureList* against, defaults to 'Feature Name'
- **message** (*str*) – append this message to the 'Exclusion Details' field for each feature excluded, defaults to 'User Excluded'

**Returns** A list of ID passed in *featureList* that could not be matched against the feature IDs present.

**Return type** list

**exportDataset** (*destinationPath*='.', *saveFormat*='CSV', *isaDetailsDict*={}, *withExclusions*=True, *escapeDelimiters*=False, *filterMetadata*=True)

Export dataset object in a variety of formats for import in other software, the export is named according to the *name* attribute of the Dataset object.

Possible save formats are:

- **CSV** Basic CSV output, *featureMetadata*, *sampleMetadata* and *intensityData* are written to three separate CSV files in *desitinationPath*
- **UnifiedCSV** Exports *featureMetadata*, *sampleMetadata* and *intensityData* concatenated into a single CSV file
- **ISATAB** Exports the sampleMetadata in the **ISATAB** format

#### Parameters

- **destinationPath** (*str*) – Save data into the directory specified here
- **format** (*str*) – File format for saved data, defaults to CSV.
- **detailsDict** (*dict*) – Contains several key: value pairs required to for exporting ISATAB.

```

detailsDict should have the format: detailsDict = {
    'investigation_identifier' : "i1", 'investigation_title' : "Give it a title", 'investigation_description' : "Add a description", 'investigation_submission_date' : "2016-11-03", 'investigation_public_release_date' : "2016-11-03", 'first_name' : "Noureddin", 'last_name' : "Sadawi", 'affiliation' : "University", 'study_filename' : "my_ms_study", 'study_material_type' : "Serum", 'study_identifier' : "s1", 'study_title' : "Give the study a title", 'study_description' : "Add study description", 'study_submission_date' : "2016-11-03", 'study_public_release_date' : "2016-11-03", 'assay_filename' : "my_ms_assay"
}

```

#### Parameters

- **withExclusions** (*bool*) – If `True` mask features and samples will be excluded
- **escapeDelimiters** (*bool*) – If `True` remove characters commonly used as delimiters in csv files from metadata
- **filterMetadata** (*bool*) – If `True` does not export the sampleMetadata and featureMetadata columns listed in `self.Attributes['sampleMetadataNotExported']` and `self.Attributes['featureMetadataNotExported']`

**Raises ValueError** – if *saveFormat* is not understood

**getFeatures** (*featureIDs*, *by=None*, *useMasks=True*)

Get a feature or list of features by name or ranges.

If `VariableType` is *Discrete*, `getFeature()` expects either a single or list of values, and matching features are returned. If `VariableType` is *Spectral*, pass either a single, or list of (min, max) tuples, the features returned will be a slice of the combined ranges. If the ranges passed overlap, the union will be returned.

#### Parameters

- **featureIDs** – A single or list of feature IDs to return
- **by** (*None* or *str*) – Column in *featureMetadata* to search in, `None` use the column defined in `Attributes['Feature Names']`

**Returns** (featureMetadata, intensityData)

**Return type** (pandas.DataFrame, numpy.ndarray)

**class** nPYc.objects.**MSDataset** (*datapath*, *fileType='QI'*, *sop='GenericMS'*, *\*\*kwargs*)

*MSDataset* extends *Dataset* to represent both peak-picked LC- or DI-MS datasets (discrete variables), and Continuum mode (spectral) DI-MS datasets.

Objects can be initialised from a variety of common data formats, currently peak-picked data from Progenesis QI or XCMS, and targeted Biocrates datasets.

- **Progenesis QI** QI import operates on csv files exported *via* the 'Export Compound Measurements' menu option in QI. Import requires the presence of both normalised and raw datasets, but will only import the raw measurements.
- **XCMS** XCMS import operates on the csv files generated by XCMS with the `peakTable()` method. By default, the csv is expected to have 14 columns of feature parameters, with the intensity values for the first sample coming on the 15 column. However, the number of columns to skip is dataset dependent and can be set with the (`noFeatureParams=` keyword argument).
- **Biocrates** Operates on spreadsheets exported from Biocrates MetIDQ. By default loads data from the sheet named 'Data Export', this may be overridden with the `sheetName=` argument, If the

number of sample metadata columns differs from the default, this can be overridden with the `noSampleParams=` argument.

**correlationToDilution**

Returns the correlation of features to dilution as calculated on samples marked as 'Dilution Series' in *sampleMetadata*, with dilution expressed in 'Dilution'.

**Returns** Vector of feature correlations to dilution

**Return type** `numpy.ndarray`

**artificialLinkageMatrix**

Gets overlapping artificial features.

**rsdSP**

Returns percentage *relative standard deviations* for each feature in the dataset, calculated on samples with the Assay Role *PrecisionReference* and Sample Type *StudyPool* in *sampleMetadata*.

**Returns** Vector of feature RSDs

**Return type** `numpy.ndarray`

**rsdSS**

Returns percentage *relative standard deviations* for each feature in the dataset, calculated on samples with the Assay Role *Assay* and Sample Type *StudySample* in *sampleMetadata*.

**Returns** Vector of feature RSDs

**Return type** `numpy.ndarray`

**applyMasks()**

Permanently delete elements masked (those set to `False`) in *sampleMask* and *featureMask*, from *featureMetadata*, *sampleMetadata*, and *intensityData*.

Resets feature linkage matrix and feature correlations.

**updateMasks** (*filterSamples=True*, *filterFeatures=True*, *sampleTypes*=[*<SampleType.StudySample>*,  
*<SampleType.StudyPool>*, *<SampleType.ExternalReference>*, *<SampleType.MethodReference>*,  
*<SampleType.ProceduralBlank>*], *assayRoles*=[*<AssayRole.Assay>*,  
*<AssayRole.PrecisionReference>*, *<AssayRole.LinearityReference>*,  
*<AssayRole.Blank>*], *featureFilters*={'artificialFilter': `False`,  
'blankFilter': `False`, 'correlationToDilutionFilter': `True`, 'rsdFilter': `True`,  
'varianceRatioFilter': `True`}, \*\*kwargs)

Update *sampleMask* and *featureMask* according to QC parameters.

*updateMasks()* sets *sampleMask* or *featureMask* to `False` for those items failing analytical criteria.

---

**Note:** To avoid reintroducing items manually excluded, this method only ever sets items to `False`, therefore if you wish to move from more stringent criteria to a less stringent set, you will need to reset the mask to all `True` using *initialiseMasks()*.

---

**Parameters**

- **filterSamples** (*bool*) – If `False` don't modify *sampleMask*
- **filterFeatures** (*bool*) – If `False` don't modify *featureMask*
- **sampleTypes** (*SampleType*) – List of types of samples to retain
- **assayRoles** (*AssayRole*) – List of assays roles to retain



- **correlationThreshold** (*None or float*) – Mask features with a correlation below this value. If *None*, use the value from *Attributes['corrThreshold']*
- **rsdThreshold** (*None or float*) – Mask features with a RSD below this value. If *None*, use the value from *Attributes['rsdThreshold']*
- **varianceRatio** (*None or float*) – Mask features where the RSD measured in study samples is below that measured in study reference samples multiplied by *varianceRatio*
- **withArtifactualFiltering** (*None or bool*) – If *None* use the value from *Attributes['artifactualFilter']*. If *False* doesn't apply artifactual filtering. If *Attributes['artifactualFilter']* is set to *False* artifactual filtering will not take place even if *withArtifactualFiltering* is set to *True*.
- **deltaMzArtifactual** (*None or float*) – Maximum allowed m/z distance between two grouped features. If *None*, use the value from *Attributes['deltaMzArtifactual']*
- **overlapThresholdArtifactual** (*None or float*) – Minimum peak overlap between two grouped features. If *None*, use the value from *Attributes['overlapThresholdArtifactual']*
- **corrThresholdArtifactual** (*None or float*) – Minimum correlation between two grouped features. If *None*, use the value from *Attributes['corrThresholdArtifactual']*
- **blankThreshold** (*None, False, or float*) – Mask features that's median intensity falls below *blankThreshold x the level in the blank*. If *False* do not filter, if *None* use the cutoff from *Attributes['blankThreshold']*, otherwise use the cutoff scaling factor provided

#### **saveFeatureMask()**

Updates featureMask and saves as 'Passing Selection' in self.featureMetadata

#### **addSampleInfo** (*descriptionFormat=None, filePath=None, filenameSpec=None, filetype='Waters.raw', \*\*kwargs*)

Load additional metadata and map it in to the *sampleMetadata* table.

Possible options:

- **'NPC LIMS'** NPC LIMS files mapping files names of raw analytical data to sample IDs
- **'NPC Subject Info'** Map subject metadata from a NPC sample manifest file (format defined in 'PC-SOP.082')
- **'Raw Data'** Extract analytical parameters from raw data files
- **'ISATAB'** ISATAB study designs
- **'Filenames'** Parses sample information out of the filenames, based on the named capture groups in the regex passed in *filenameSpec*
- **'Basic CSV'** Joins the *sampleMetadata* table with the data in the *csv* file at *filePath=*, matching on the 'Sample File Name' column in both.

#### **Parameters**

- **descriptionFormat** (*str*) – Format of metadata to be added
- **filePath** (*str*) – Path to the additional data to be added
- **filenameSpec** (*None or str*) – Only used if *descriptionFormat* is 'Filenames'. A regular expression that extracts sample-type information into the following named capture

groups: 'fileName', 'baseName', 'study', 'chromatography' 'ionisation', 'instrument', 'groupingKind' 'groupingNo', 'injectionKind', 'injectionNo', 'reference', 'exclusion' 're-runs', 'extraInjections', 'exclusion2'. if `None` is passed, use the *filenameSpec* key in *Attributes*, loaded from the SOP json

**Raises** `NotImplementedError` – if the *descriptionFormat* is not understood

**amendBatches** (*sampleRunOrder*)

Creates a new batch starting at the sample index in *sampleRunOrder*, and amends subsequent batch numbers in *sampleMetadata*['Correction Batch']

**Parameters** *sampleRunOrder* (*int*) – Index of first sample in new batch

**artifactualFilter** (*featMask=None*)

Filter artifactual features on top of the *featureMask* already present if none given as input Keep feature with the highest intensity on the mean spectra

**Parameters** *featMask* (*numpy.ndarray* or *None*) – A *featureMask* (True for inclusion), if *None*, use *featureMask*

**Returns** Amended *featureMask*

**Return type** *numpy.ndarray*

**excludeFeatures** (*featureList*, *on='Feature Name'*, *message='User Excluded'*)

Masks the features listed in *featureList* from the dataset.

**Parameters**

- **featureList** (*list*) – A list of feature IDs to be excluded
- **on** (*str*) – name of the column in *featureMetadata* to match *featureList* against, defaults to 'Feature Name'
- **message** (*str*) – append this message to the 'Exclusion Details' field for each feature excluded, defaults to 'User Excluded'

**Returns** A list of ID passed in *featureList* that could not be matched against the feature IDs present.

**Return type** *list*

**initialiseMasks** ()

Re-initialise *featureMask* and *sampleMask* to match the current dimensions of *intensityData*, and include all samples.

**validateObject** (*verbose=True*, *raiseError=False*, *raiseWarning=True*)

Checks that all the attributes specified in the class definition are present and of the required class and/or values.

Returns 4 boolean: is the object a *Dataset* < a *basic MSDataSet* < has the object *parameters for QC* < has the object *sample metadata*.

To employ all class methods, the most inclusive (*has the object sample metadata*) must be successful:

- 'Basic MSDataSet' checks *Dataset* types and uniqueness as well as additional attributes.
- 'has parameters for QC' is 'Basic MSDataSet' + *sampleMetadata*[['SampleType', 'AssayRole', 'Dilution', 'Run Order', 'Batch', 'Correction Batch', 'Sample Base Name']]
- 'has sample metadata' is 'has parameters for QC' + *sampleMetadata*[['Sample ID', 'Subject ID', 'Matrix']]

Column type() in pandas.DataFrame are established on the first sample when necessary Does not check for uniqueness in sampleMetadata['Sample File Name'] Does not currently check Attributes['Raw Data Path'] type Does not currently check corrExclusions type

#### Parameters

- **verbose** (*bool*) – if True the result of each check is printed (default True)
- **raiseError** (*bool*) – if True an error is raised when a check fails and the validation is interrupted (default False)
- **raiseWarning** (*bool*) – if True a warning is raised when a check fails

**Returns** A dictionary of 4 boolean with True if the Object conforms to the corresponding test. 'Dataset' conforms to *Dataset*, 'BasicMSDataset' conforms to *Dataset* + basic *MSDataset*, 'QC' BasicMSDataset + object has QC parameters, 'sampleMetadata' QC + object has sample metadata information

**Return type** dict

#### Raises

- **TypeError** – if the Object class is wrong
- **AttributeError** – if self.Attributes['rtWindow'] does not exist
- **TypeError** – if self.Attributes['rtWindow'] is not an int or float
- **AttributeError** – if self.Attributes['msPrecision'] does not exist
- **TypeError** – if self.Attributes['msPrecision'] is not an int or float
- **AttributeError** – if self.Attributes['varianceRatio'] does not exist
- **TypeError** – if self.Attributes['varianceRatio'] is not an int or float
- **AttributeError** – if self.Attributes['blankThreshold'] does not exist
- **TypeError** – if self.Attributes['blankThreshold'] is not an int or float
- **AttributeError** – if self.Attributes['corrMethod'] does not exist
- **TypeError** – if self.Attributes['corrMethod'] is not a str
- **AttributeError** – if self.Attributes['corrThreshold'] does not exist
- **TypeError** – if self.Attributes['corrThreshold'] is not an int or float
- **AttributeError** – if self.Attributes['rsdThreshold'] does not exist
- **TypeError** – if self.Attributes['rsdThreshold'] is not an int or float
- **AttributeError** – if self.Attributes['artifactualFilter'] does not exist
- **TypeError** – if self.Attributes['artifactualFilter'] is not a bool
- **AttributeError** – if self.Attributes['deltaMzArtifactual'] does not exist
- **TypeError** – if self.Attributes['deltaMzArtifactual'] is not an int or float
- **AttributeError** – if self.Attributes['overlapThresholdArtifactual'] does not exist
- **TypeError** – if self.Attributes['overlapThresholdArtifactual'] is not an int or float
- **AttributeError** – if self.Attributes['corrThresholdArtifactual'] does not exist
- **TypeError** – if self.Attributes['corrThresholdArtifactual'] is not an int or float
- **AttributeError** – if self.Attributes['FeatureExtractionSoftware'] does not exist

- **TypeError** – if self.Attributes['FeatureExtractionSoftware'] is not a str
- **AttributeError** – if self.Attributes['Raw Data Path'] does not exist
- **TypeError** – if self.Attributes['Raw Data Path'] is not a str
- **AttributeError** – if self.Attributes['Feature Names'] does not exist
- **TypeError** – if self.Attributes['Feature Names'] is not a str
- **TypeError** – if self.VariableType is not an enum 'VariableType'
- **AttributeError** – if self.corrExclusions does not exist
- **AttributeError** – if self.\_correlationToDilution does not exist
- **TypeError** – if self.\_correlationToDilution is not a numpy.ndarray
- **AttributeError** – if self.\_artifactualLinkageMatrix does not exist
- **TypeError** – if self.\_artifactualLinkageMatrix is not a pandas.DataFrame
- **AttributeError** – if self.\_tempArtifactualLinkageMatrix does not exist
- **TypeError** – if self.\_tempArtifactualLinkageMatrix is not a pandas.DataFrame
- **AttributeError** – if self.fileName does not exist
- **TypeError** – if self.fileName is not a str
- **AttributeError** – if self.filePath does not exist
- **TypeError** – if self.filePath is not a str
- **ValueError** – if self.sampleMetadata does not have the same number of samples as self.\_intensityData
- **TypeError** – if self.sampleMetadata['Sample File Name'] is not str
- **TypeError** – if self.sampleMetadata['AssayRole'] is not an enum 'AssayRole'
- **TypeError** – if self.sampleMetadata['SampleType'] is not an enum 'SampleType'
- **TypeError** – if self.sampleMetadata['Dilution'] is not an int or float
- **TypeError** – if self.sampleMetadata['Batch'] is not an int or float
- **TypeError** – if self.sampleMetadata['Correction Batch'] is not an int or float
- **TypeError** – if self.sampleMetadata['Run Order'] is not an int
- **TypeError** – if self.sampleMetadata['Acquired Time'] is not a datetime
- **TypeError** – if self.sampleMetadata['Sample Base Name'] is not str
- **LookupError** – if self.sampleMetadata does not have a Matrix column
- **TypeError** – if self.sampleMetadata['Matrix'] is not a str
- **LookupError** – if self.sampleMetadata does not have a Subject ID column
- **TypeError** – if self.sampleMetadata['Subject ID'] is not a str
- **TypeError** – if self.sampleMetadata['Sample ID'] is not a str
- **ValueError** – if self.featureMetadata does not have the same number of features as self.\_intensityData
- **TypeError** – if self.featureMetadata['Feature Name'] is not a str
- **ValueError** – if self.featureMetadata['Feature Name'] is not unique

- **LookupError** – if self.featureMetadata does not have a m/z column
- **TypeError** – if self.featureMetadata['m/z'] is not an int or float
- **LookupError** – if self.featureMetadata does not have a Retention Time column
- **TypeError** – if self.featureMetadata['Retention Time'] is not an int or float
- **ValueError** – if self.sampleMask has not been initialised
- **ValueError** – if self.sampleMask does not have the same number of samples as self.\_intensityData
- **ValueError** – if self.featureMask has not been initialised
- **ValueError** – if self.featureMask does not have the same number of features as self.\_intensityData

**class** nPYc.objects.NMRDataset (datapath, fileType='Bruker', sop='GenericNMRurine', pulseprogram='noesygppld', \*\*kwargs)

*NMRDataset* extends *Dataset* to represent both spectral and peak-picked NMR datasets.

Objects can be initialised from a variety of common data formats, including Bruker-format raw data, and BI-LISA targeted lipoprotein analysis.

- **Bruker** When loading Bruker format raw spectra (1r files), all directories below *datapath* will be scanned for valid raw data, and those matching *pulseprogram* loaded and aligned onto a common scale as defined in *sop*.
- **BI-LISA** BI-LISA data can be read from Excel workbooks, the name of the sheet containing the data to be loaded should be passed in the *pulseProgram* argument. Feature descriptors will be loaded from the 'Analytes' sheet, and file names converted back to the *ExperimentName/expno* format from *ExperimentName\_EXPNO\_expno*.

#### Parameters

- **fileType** (*str*) – Type of data to be loaded
- **sheetname** (*str*) – Load data from the specified sheet of the Excel workbook
- **pulseprogram** (*str*) – When loading raw data, only import spectra acquired with *pulseprogram*

**addSampleInfo** (*descriptionFormat=None, filePath=None, filenameSpec=None, \*\*kwargs*)

Load additional metadata and map it in to the *sampleMetadata* table.

Possible options:

- **'NPC LIMS'** NPC LIMS files mapping file names of raw analytical data to sample IDs
- **'NPC Subject Info'** Map subject metadata from a NPC sample manifest file (format defined in 'PC-SOP.082')
- **'Raw Data'** Extract analytical parameters from raw data files
- **'ISATAB'** ISATAB study designs
- **'Filenames'** Parses sample information out of the filenames, based on the named capture groups in the regex passed in *filenameSpec*
- **'Basic CSV'** Joins the *sampleMetadata* table with the data in the *csv* file at *filePath=*, matching on the 'Sample File Name' column in both.

#### Parameters

- **descriptionFormat** (*str*) – Format of metadata to be added
- **filePath** (*str*) – Path to the additional data to be added
- **filenameSpec** (*None or str*) – Only used if *descriptionFormat* is 'FileNames'. A regular expression that extracts sample-type information into the following named capture groups: 'fileName', 'baseName', 'study', 'chromatography', 'ionisation', 'instrument', 'groupingKind', 'groupingNo', 'injectionKind', 'injectionNo', 'reference', 'exclusion', 're-runs', 'extraInjections', 'exclusion2'. if *None* is passed, use the *filenameSpec* key in *Attributes*, loaded from the SOP json

**Raises** `NotImplementedError` – if the *descriptionFormat* is not understood

**updateMasks** (*filterSamples=True, filterFeatures=True, sampleTypes=[<SampleType.StudySample>, <SampleType.StudyPool>, <SampleType.ExternalReference>, <SampleType.MethodReference>, <SampleType.ProceduralBlank>], assayRoles=[<AssayRole.Assay>, <AssayRole.PrecisionReference>, <AssayRole.LinearReference>, <AssayRole.Blank>], exclusionRegions=None, sampleQCChecks=[], \*\*kwargs)*

Update *sampleMask* and *featureMask* according to parameters.

*updateMasks()* sets *sampleMask* or *featureMask* to `False` for those items failing analytical criteria.

---

**Note:** To avoid reintroducing items manually excluded, this method only ever sets items to `False`, therefore if you wish to move from more stringent criteria to a less stringent set, you will need to reset the mask to all `True` using *initialiseMasks()*.

---

#### Parameters

- **filterSamples** (*bool*) – If `False` don't modify *sampleMask*
- **filterFeatures** (*bool*) – If `False` don't modify *featureMask*
- **sampleTypes** (*SampleType*) – List of types of samples to retain
- **sampleRoles** (*AssayRole*) – List of assays roles to retain
- **exclusionRegions** (*list of tuple*) – If `None` Exclude ranges defined in *Attributes*['exclusionRegions']
- **sampleQCChecks** (*list*) – Which quality control metrics to use.

**plot** (*spectra, labels, interactive=False*)

Plots a set of nmr spectra. If *interactive* is `False`, returns a static matplotlib plot. If `True`, then *plotly* is used to generate an interactive plot.

#### Parameters

- **spectra** – The specific 'labels' of the spectra to plot. By default all spectra are plotted.
- **labels** – Which labels to select
- **interactive** – Use matplotlib (`False`) or *plotly* (`True`)

**Returns** Displays the NMR data and returns either a matplotlib axis object or a *plotly* figure dictionary

```
class nPYc.objects.TargetedDataset (dataPath,      fileType='TargetLynx',      sop='Generic',
                                   **kwargs)
```

*TargetedDataset* extends *Dataset* to represent quantitative datasets, where compounds are already identified, the exactitude of the quantification can be established, units are known and calibration curve or internal standards are employed. The *TargetedDataset* class include methods to apply limits of quantification (LLOQ and ULOQ), merge multiple analytical batch, and report accuracy and precision of each measurements.

In addition to the structure of *Dataset*, *TargetedDataset* requires the following attributes:

- **expectedConcentration:** A  $n \times m$  pandas dataframe of expected concentrations (matching the *intensityData* dimension), with column names matching `featureMetadata['Feature Name']`
- **calibration:** A dictionary containing pandas dataframe describing calibration samples:
  - **calibration['calibIntensityData']:** A  $r \times m$  numpy matrix of measurements. Features must match features in *intensityData*
  - **calibration['calibSampleMetadata']:** A  $r \times m$  pandas dataframe of calibration sample identifiers and metadata
  - **calibration['calibFeatureMetadata']:** A  $m \times q$  pandas dataframe of feature identifiers and metadata
  - **calibration['calibExpectedConcentration']:** A  $r \times m$  pandas dataframe of calibration samples expected concentrations
- **Attributes** must contain the following (can be loaded from a method specific JSON on import):
  - **methodName:** A (str) name of the method
  - **externalID:** A list of external ID, each external ID must also be present in *Attributes* as a list of identifier (for that external ID) for each feature. For example, if `externalID=['PubChem ID']`, `Attributes['PubChem ID']=['ID1', 'ID2', '', 'ID75']`
- **featureMetadata** expects the following columns:
  - **quantificationType:** A *QuantificationType* enum specifying the exactitude of the quantification procedure employed.
  - **calibrationMethod:** A *CalibrationMethod* enum specifying the calibration method employed.
  - **Unit:** A (str) unit corresponding the the feature measurement value.
  - **LLOQ:** The lowest limit of quantification, used to filter concentrations  $< \text{LLOQ}$
  - **ULOQ:** The upper limit of quantification, used to filter concentrations  $> \text{ULOQ}$
  - **externalID:** All externalIDs listed in `Attributes['externalID']` must be present as their own column

Currently targeted assay results processed using **TargetLynx** or **Bruker quantification results** can be imported. To create an import for any other form of semi-quantitative or quantitative results, the procedure is as follow:

- Create a new `fileType == 'myMethod'` entry in `__init__()`
- Define functions to populate all expected dataframes (using file readers, JSON,...)
- Separate calibration samples from study samples (store in `calibration`). *If none exist, initialise empty dataframes with the correct number of columns and column names.*
- Execute pre-processing steps if required (note: all feature values should be expressed in the unit listed in `featureMetadata['Unit']`)

- Apply limits of quantification using `_applyLimitsOfQuantification()`. (This function does not apply limits of quantification to features marked as `QuantificationType == QuantificationType.Monitored` for compounds monitored for relative information.)

The resulting `TargetedDataset` created must satisfy to the criteria for *BasicTargetedDataset*, which can be checked with `validatedObject()` (list the minimum requirements for all class methods).

- `fileType == 'TargetLynx'` to import data processed using **TargetLynx**

TargetLynx import operates on xml files exported *via* the 'File -> Export -> XML' TargetLynx menu option. Import requires a `calibration_report.csv` providing lower and upper limits of quantification (LLOQ, ULOQ) with the `calibrationReportPath` keyword argument.

Targeted data measurements as well as calibration report information are read and mapped with pre-defined SOPs. All measurements are converted to pre-defined units and measurements inferior to the lowest limits of quantification or superior to the upper limits of quantification are replaced. Once the import is finished, only analysed samples are returned (no calibration samples) and only features mapped onto the pre-defined SOP and sufficiently described.

Instructions to create new TargetLynx SOP can be found on the [generation of targeted SOPs](#) page.

Example: 

```
TargetedDataset(datapath, fileType='TargetLynx',
sop='OxylipinMS', calibrationReportPath=calibrationReportPath,
sampleTypeToProcess=['Study Sample','QC'], noiseFilled=False,
onlyLLOQ=False, responseReference=None)
```

- **sop** Currently implemented are 'OxylipinMS' and 'AminoAcidMS'

*AminoAcidMS*: Gray N. *et al.* Human Plasma and Serum via Precolumn Derivatization with 6-Aminoquinolyl-N-hydroxysuccinimidyl Carbamate: Application to Acetaminophen-Induced Liver Failure. *Analytical Chemistry*, 2017, 89, 247887.

*OxylipinMS*: Wolfer AM. *et al.* Development and Validation of a High-Throughput Ultrahigh-Performance Liquid Chromatography-Mass Spectrometry Approach for Screening of Oxylipins and Their Precursors. *Analytical Chemistry*, 2015, 87 (23),11721–31

- **calibrationReportPath** Path to the calibration report *csv* following the provided report template.

The following columns are required (leave an empty value to reject a compound):

- \* **Compound** The compound name, identical to the one employed in the SOP *json* file.
- \* **TargetLynx ID** The compound TargetLynx ID, identical to the one employed in the SOP *json* file.
- \* **LLOQ** Lowest limit of quantification concentration, in the same unit as indicated in TargetLynx.
- \* **ULOQ** Upper limit of quantification concentration, in the same unit as indicated in TargetLynx.

The following columns are expected by `_targetLynxApplyLimitsOfQuantificationNoiseFilled`

- \* **Noise (area)** Area integrated in a blank sample at the same retention time as the compound of interest (if left empty noise concentration calculation cannot take place).
- \* **a** *a* coefficient in the calibration equation (if left empty noise concentration calculation cannot take place).



- \* **b** *b* coefficient in the calibration equation (if left empty noise concentration calculation cannot take place).

The following columns are recommended but not expected:

- \* **Cpd Info** Additional information relating to the compound (can be left empty).
- \* **r** *r* goodness of fit measure for the calibration equation (can be left empty).
- \* **r2**  $r^2$  goodness of fit measure for the calibration equation (can be left empty).
- **sampleTypeToProcess** List of ['Study Sample', 'Blank', 'QC', 'Other'] for the sample types to process as defined in MassLynx. Only samples in 'sampleTypeToProcess' are returned. Calibrants should not be processed and are not returned. Most uses should only require 'Study Sample' as quality controls are identified based on sample names by subsequent functions. *Default value is ['Study Sample', 'QC']*.
- **noiseFilled** If True values <LLOQ will be replaced by a concentration equivalent to the noise level in a blank. If False <LLOQ is replaced by *-inf*. *Default value is 'False'*
- **onlyLLOQ** If True only correct <LLOQ, if False correct <LLOQ and >ULOQ. *Default value is 'False'*.
- **responseReference** If noiseFilled=True the noise concentration needs to be calculated. Provide the 'Sample File Name' of a reference sample to use in order to establish the response to use, or list of samples to use (one per feature). If None, the middle of the calibration will be employed. *Default value is 'None'*.
- **keepPeakInfo** If keepPeakInfo=True (default *False*) adds the peakInfo dictionary to the calibration. peakInfo contains the *peakResponse*, *peakArea*, *peakConcentrationDeviation*, *peakIntegrationFlag* and *peakRT*.
- **keepExcluded** If keepExcluded=True (default *False*), import exclusions (excludedImportSampleMetadata, excludedImportFeatureMetadata, excludedImportIntensityData and excludedImportExpectedConcentration) are kept in the object.
- **keepIS** If keepIS=True (default *False*), features marked as Internal Standards (IS) are retained.
- **fileType = 'Bruker Quantification'** to import Bruker quantification results
  - **nmrRawDataPath** Path to the parent folder where all result files are stored. All sub-folders will be parsed and the .xml results files matching the *fileNamePattern* imported.
  - **fileNamePattern** Regex to recognise the result data xml files
  - **pdata** To select the right pdata folders (default 1)

Two form of Bruker quantification results are supported and selected using the *sop* option: *BrukerQuant-UR* and *Bruker BI-LISA*

- **sop = 'BrukerQuant-UR'**

```
Example: TargetedDataset(nmrRawDataPath,
fileType='Bruker Quantification',
sop='BrukerQuant-UR', fileNamePattern='.*?
urine_quant_report_b\.xml$', unit='mmol/mol Crea')
```

- \* **unit** If features are duplicated with different units, unit limits the import to features matching said unit. (In case of duplication and no unit, all available units will be listed)

```
- sop = 'BrukerBI-LISA' Example: TargetedDataset(nmrRawDataPath,  
fileType='Bruker Quantification', sop='BrukerBI-LISA',  
fileNamePattern='.*?results\.xml$')
```

**rsdSP**

Returns percentage *relative standard deviations* for each feature in the dataset, calculated on samples with the Assay Role *PrecisionReference* and Sample Type *StudyPool* in *sampleMetadata*. Implemented as a back-up to *accuracyPrecision()* when no expected concentrations are known

**Returns** Vector of feature RSDs

**Return type** numpy.ndarray

**rsdSS**

Returns percentage *relative standard deviations* for each feature in the dataset, calculated on samples with the Assay Role *Assay* and Sample Type *StudySample* in *sampleMetadata*.

**Returns** Vector of feature RSDs

**Return type** numpy.ndarray

**mergeLimitsOfQuantification** (*keepBatchLOQ=False, onlyLLOQ=False*)

Update limits of quantification and apply LLOQ/ULOQ using the lowest common denominator across all batch (after a `__add__()`). Keep the highest LLOQ and lowest ULOQ.

**Parameters**

- **keepBatchLOQ** (*bool*) – If True do not remove each batch LOQ (`featureMetadata['LLOQ_batchX']`, `featureMetadata['ULOQ_batchX']`)
- **onlyLLOQ** (*bool*) – if True only correct <LLOQ, if False correct <LLOQ and >ULOQ

**Raises**

- **ValueError** – if targetedData does not satisfy to the BasicTargetedDataset definition on input
- **ValueError** – if number of batch, LLOQ\_batchX and ULOQ\_batchX do not match
- **ValueError** – if targetedData does not satisfy to the BasicTargetedDataset definition after LOQ merging
- **Warning** – if `featureMetadata['LLOQ']` or `featureMetadata['ULOQ']` already exist and will be overwritten.

**exportDataset** (*destinationPath='.', saveFormat='CSV', withExclusions=True, escapeDelimiters=False, filterMetadata=True*)

Calls *exportDataset()* and raises a warning if normalisation is employed as *TargetedDataset* intensityData can be left-censored.

**validateObject** (*verbose=True, raiseError=False, raiseWarning=True*)

Checks that all the attributes specified in the class definition are present and of the required class and/or values.

Returns 4 boolean: is the object a *Dataset* < a basic *TargetedDataset* < has the object parameters for *QC* < has the object sample metadata.

To employ all class methods, the most inclusive (*has the object sample metadata*) must be successful:

- ‘Basic TargetedDataset’ checks *TargetedDataset* types and uniqueness as well as additional attributes.

- *'has parameters for QC'* is *'Basic TargetedDataset'* + sampleMetadata[['SampleType, AssayRole, Dilution, Run Order, Batch, Correction Batch, Sample Base Name]]
- *'has sample metadata'* is *'has parameters for QC'* + sampleMetadata[['Sample ID', 'Subject ID', 'Matrix']]

calibration['calibIntensityData'] must be initialised even if no samples are present calibration['calibSampleMetadata'] must be initialised even if no samples are present, use: pandas.DataFrame(None, columns=self.sampleMetadata.columns.values.tolist()) calibration['calibFeatureMetadata'] must be initialised even if no samples are present, use a copy of self.featureMetadata calibration['calibExpectedConcentration'] must be initialised even if no samples are present, use: pandas.DataFrame(None, columns=self.expectedConcentration.columns.values.tolist()) Calibration features must be identical to the usual features. Number of calibration samples and features must match across the 4 calibration tables If *'sampleMetadataExcluded'*, *'intensityDataExcluded'*, *'featureMetadataExcluded'*, *'expectedConcentrationExcluded'* or *'excludedFlag'* exist, the existence and number of exclusions (based on *'sampleMetadataExcluded'*) is checked

Column type() in pandas.DataFrame are established on the first sample (for non int/float) featureMetadata are search for column names containing 'LLOQ' & 'ULOQ' to allow for 'LLOQ\_batch...' after \_\_add\_\_(), the first column matching is then checked for dtype If datasets are merged, calibration is a list of dict, and number of features is only kept constant inside each dict Does not check for uniqueness in sampleMetadata['Sample File Name'] Does not check columns inside calibration['calibSampleMetadata'] Does not check columns inside calibration['calibFeatureMetadata'] Does not currently check for Attributes['Feature Name']

#### Parameters

- **verbose** (*bool*) – if True the result of each check is printed (default True)
- **raiseError** (*bool*) – if True an error is raised when a check fails and the validation is interrupted (default False)
- **raiseWarning** (*bool*) – if True a warning is raised when a check fails

**Returns** A dictionary of 4 boolean with True if the Object conforms to the corresponding test. 'Dataset' conforms to *Dataset*, 'BasicTargetedDataset' conforms to *Dataset* + basic *TargetedDataset*, 'QC' BasicTargetedDataset + object has QC parameters, 'sampleMetadata' QC + object has sample metadata information

**Return type** dict

#### Raises

- **TypeError** – if the Object class is wrong
- **AttributeError** – if self.Attributes['methodName'] does not exist
- **TypeError** – if self.Attributes['methodName'] is not a str
- **AttributeError** – if self.Attributes['externalID'] does not exist
- **TypeError** – if self.Attributes['externalID'] is not a list
- **TypeError** – if self.VariableType is not an enum 'VariableType'
- **AttributeError** – if self.fileName does not exist
- **TypeError** – if self.fileName is not a str or list
- **AttributeError** – if self.filePath does not exist

- **TypeError** – if self.filePath is not a str or list
- **ValueError** – if self.sampleMetadata does not have the same number of samples as self.\_intensityData
- **TypeError** – if self.sampleMetadata['Sample File Name'] is not str
- **TypeError** – if self.sampleMetadata['AssayRole'] is not an enum 'AssayRole'
- **TypeError** – if self.sampleMetadata['SampleType'] is not an enum 'SampleType'
- **TypeError** – if self.sampleMetadata['Dilution'] is not an int or float
- **TypeError** – if self.sampleMetadata['Batch'] is not an int or float
- **TypeError** – if self.sampleMetadata['Correction Batch'] is not an int or float
- **TypeError** – if self.sampleMetadata['Run Order'] is not an int
- **TypeError** – if self.sampleMetadata['Acquired Time'] is not a datetime
- **TypeError** – if self.sampleMetadata['Sample Base Name'] is not str
- **LookupError** – if self.sampleMetadata does not have a Subject ID column
- **TypeError** – if self.sampleMetadata['Subject ID'] is not a str
- **TypeError** – if self.sampleMetadata['Sample ID'] is not a str
- **ValueError** – if self.featureMetadata does not have the same number of features as self.\_intensityData
- **TypeError** – if self.featureMetadata['Feature Name'] is not a str
- **ValueError** – if self.featureMetadata['Feature Name'] is not unique
- **LookupError** – if self.featureMetadata does not have a calibrationMethod column
- **TypeError** – if self.featureMetadata['calibrationMethod'] is not an enum 'CalibrationMethod'
- **LookupError** – if self.featureMetadata does not have a quantificationType column
- **TypeError** – if self.featureMetadata['quantificationType'] is not an enum 'QuantificationType'
- **LookupError** – if self.featureMetadata does not have a Unit column
- **TypeError** – if self.featureMetadata['Unit'] is not a str
- **LookupError** – if self.featureMetadata does not have a LLOQ or similar column
- **TypeError** – if self.featureMetadata['LLOQ'] or similar is not an int or float
- **LookupError** – if self.featureMetadata does not have a ULOQ or similar column
- **TypeError** – if self.featureMetadata['ULOQ'] or similar is not an int or float
- **LookupError** – if self.featureMetadata does not have the 'externalID' as columns
- **AttributeError** – if self.expectedConcentration does not exist
- **TypeError** – if self.expectedConcentration is not a pandas.DataFrame
- **ValueError** – if self.expectedConcentration does not have the same number of samples as self.\_intensityData
- **ValueError** – if self.expectedConcentration does not have the same number of features as self.\_intensityData

- **ValueError** – if self.expectedConcentration column name do not match self.featureMetadata['Feature Name']
- **ValueError** – if self.sampleMask is not initialised
- **ValueError** – if self.sampleMask does not have the same number of samples as self.\_intensityData
- **ValueError** – if self.featureMask has not been initialised
- **ValueError** – if self.featureMask does not have the same number of features as self.\_intensityData
- **AttributeError** – if self.calibration does not exist
- **TypeError** – if self.calibration is not a dict
- **AttributeError** – if self.calibration['calibIntensityData'] does not exist
- **TypeError** – if self.calibration['calibIntensityData'] is not a numpy.ndarray
- **ValueError** – if self.calibration['calibIntensityData'] does not have the same number of features as self.\_intensityData
- **AttributeError** – if self.calibration['calibSampleMetadata'] does not exist
- **TypeError** – if self.calibration['calibSampleMetadata'] is not a pandas.DataFrame
- **ValueError** – if self.calibration['calibSampleMetadata'] does not have the same number of samples as self.calibration['calibIntensityData']
- **AttributeError** – if self.calibration['calibFeatureMetadata'] does not exist
- **TypeError** – if self.calibration['calibFeatureMetadata'] is not a pandas.DataFrame
- **LookupError** – if self.calibration['calibFeatureMetadata'] does not have a ['Feature Name'] column
- **ValueError** – if self.calibration['calibFeatureMetadata'] does not have the same number of features as self.\_intensityData
- **AttributeError** – if self.calibration['calibExpectedConcentration'] does not exist
- **TypeError** – if self.calibration['calibExpectedConcentration'] is not a pandas.DataFrame
- **ValueError** – if self.calibration['calibExpectedConcentration'] does not have the same number of samples as self.calibration['calibIntensityData']
- **ValueError** – if self.calibration['calibExpectedConcentration'] does not have the same number of features as self.calibration['calibIntensityData']
- **ValueError** – if self.calibration['calibExpectedConcentration'] column name do not match self.featureMetadata['Feature Name']

#### **applyMasks()**

Permanently delete elements masked (those set to False) in *sampleMask* and *featureMask*, from *featureMetadata*, *sampleMetadata*, *intensityData* and *py:attr:TargetedDataset.expectedConcentration*.

Features are excluded in each calibration based on the internal `calibration['calibFeatureMetadata']` (iterate through the list of calibration if 2+ datasets have been joined with `__add__()`).

```
updateMasks (filterSamples=True, filterFeatures=True, sampleTypes=[<SampleType.StudySample>,
<SampleType.StudyPool>], assayRoles=[<AssayRole.Assay>, <Assay-
Role.PrecisionReference>], quantificationTypes=[<QuantificationType.IS>,
<QuantificationType.QuantOwnLabeledAnalogue>, <Quantification-
Type.QuantAltLabeledAnalogue>, <QuantificationType.QuantOther>, <Quantifica-
tionType.Monitored>], calibrationMethods=[<CalibrationMethod.backcalculatedIS>,
<CalibrationMethod.noIS>, <CalibrationMethod.noCalibration>, <Calibration-
Method.otherCalibration>], rsdThreshold=None, **kwargs)
```

Update *sampleMask* and *featureMask* according to QC parameters.

*updateMasks()* sets *sampleMask* or *featureMask* to False for those items failing analytical criteria.

Similar to *updateMasks()*, without *blankThreshold* or *artifactual* filtering

---

**Note:** To avoid reintroducing items manually excluded, this method only ever sets items to False, therefore if you wish to move from more stringent criteria to a less stringent set, you will need to reset the mask to all True using *initialiseMasks()*.

---

### Parameters

- **filterSamples** (*bool*) – If False don't modify sampleMask
- **filterFeatures** (*bool*) – If False don't modify featureMask
- **sampleTypes** (*SampleType*) – List of types of samples to retain
- **assayRoles** (*AssayRole*) – List of assays roles to retain
- **quantificationTypes** (*QuantificationType*) – List of quantification types to retain
- **calibrationMethods** (*CalibrationMethod*) – List of calibration methods to retain

### Raises

- **TypeError** – if sampleTypes is not a list
- **TypeError** – if sampleTypes are not a SampleType enum
- **TypeError** – if assayRoles is not a list
- **TypeError** – if assayRoles are not an AssayRole enum
- **TypeError** – if quantificationTypes is not a list
- **TypeError** – if quantificationTypes are not a QuantificationType enum
- **TypeError** – if calibrationMethods is not a list
- **TypeError** – if calibrationMethods are not a CalibrationMethod enum

```
addSampleInfo (descriptionFormat=None, filePath=None, **kwargs)
```

Load additional metadata and map it in to the *sampleMetadata* table.

Possible options:

- **'NPC Subject Info'** Map subject metadata from a NPC sample manifest file (format defined in 'PCSOP.082')
- **'Raw Data'** Extract analytical parameters from raw data files

- **‘ISATAB’** ISATAB study designs
- **‘FileNames’** Parses sample information out of the filenames, based on the named capture groups in the regex passed in *filenameSpec*
- **‘Basic CSV’** Joins the `sampleMetadata` table with the data in the `csv` file at *filePath*=, matching on the ‘Sample File Name’ column in both.
- **‘Batches’** Interpolate batch numbers for samples between those with defined batch numbers based on sample acquisitions times

#### Parameters

- **descriptionFormat** (*str*) – Format of metadata to be added
- **filePath** (*str*) – Path to the additional data to be added
- **filenameSpec** (*None or str*) – Only used if *descriptionFormat* is ‘FileNames’. A regular expression that extracts sample-type information into the following named capture groups: ‘fileName’, ‘baseName’, ‘study’, ‘chromatography’ ‘ionisation’, ‘instrument’, ‘groupingKind’ ‘groupingNo’, ‘injectionKind’, ‘injectionNo’, ‘reference’, ‘exclusion’ ‘reruns’, ‘extraInjections’, ‘exclusion2’. if *None* is passed, use the *filenameSpec* key in *Attributes*, loaded from the SOP json

**Raises** `NotImplementedError` – if the *descriptionFormat* is not understood

**accuracyPrecision** (*onlyPrecisionReferences=False*)

Return Precision (percent RSDs) and Accuracy for each SampleType and each unique concentration. Statistic grouped by SampleType, Feature and unique concentration.

#### Parameters

- **dataset** (`TargetedDataset`) – `TargetedDataset` object to generate the accuracy and precision for.
- **onlyPrecisionReference** (*bool*) – If `True` only use samples with the *Assay-Role* `PrecisionReference`.

**Returns** Dict of Accuracy and Precision dict for each group.

**Return type** dict(str:dict(str:pandas.DataFrame))

**Raises** `TypeError` – if dataset is not an instance of `TargetedDataset`





---

## Sample Metadata

---

Using the nPYc-Toolbox, additional study design parameters or sample metadata may be mapped into the Dataset using the `addSampleInfo()` method. This additional sample information may be added from a number of different sources, for example, from an associated CSV file, or from the raw data (see below), ‘addSampleInfo’ extracts the appropriate information, matches it to the acquired samples, and adds it into the `sampleMetadata` attribute of the dataset (a pandas dataframe of sample identifiers and sample associated metadata (see *Datasets* for details).

### 6.1 CSV Template for Metadata Import

The ‘Basic CSV’ format specifies a simple method for matching analytical data to metadata using:

```
dataset.addSampleInfo(descriptionFormat='Basic CSV', filePath='path to basicCSV.csv')
```

Although optional, it is recommend to generate such a CSV file containing basic metadata about each of the imported spectra.

The nPYc-Toolbox options contains a default syntax for adding sample metadata in a predefined CSV format.

In brief, this CSV file format expects information to be provided for 6 pre-defined column names, ‘Sample File Name’, ‘Sample ID’, ‘SampleType’, ‘AssayRole’, ‘Dilution’, ‘Include Sample’. Any extra metadata (such as patient characteristics or clinical metadata) can be placed in this file, as long as the column names are not in the list of expected fields.

- ‘Sample ID’: Unique identifier for each sample
- ‘Sample File Name’: the ‘Basic CSV’ file matches based on the entries in the ‘Sample File Name’ column to the ‘Sample File Name’ in the `sampleMetadata` table
- ‘AssayRole’: *assay role* as described in *Recommended Study Design Elements*
- ‘SampleType’: *sample type* as described in *Recommended Study Design Elements*
- ‘Dilution’: Relative dilution factor for each sample
- ‘Include Sample’: where ‘Include Sample’ is `False`, the `sampleMask` for that sample will be set to `False` and the corresponding sample marked for exclusion from the dataset (see *Sample and Feature Masks* for details)

Table 1: Minimal structure of a basic csv file

Sample ID	Sample File Name	AssayRole	SampleType	Dilution	Include Sample
Dilution 1	UnitTest1_LPOS_ToF02_B1SRD01	Linearity Reference	Study Pool	1	TRUE
Dilution 2	UnitTest1_LPOS_ToF02_B1SRD02	Linearity Reference	Study Pool	50	TRUE
Sample 1	UnitTest1_LPOS_ToF02_S1W07	Assay	Study Sample	100	TRUE
Sample 2	UnitTest1_LPOS_ToF02_S1W08	Assay	Study Sample	100	TRUE
LTR	UnitTest1_LPOS_ToF02_S1W11	Precision Reference	External Reference	100	TRUE
SR	UnitTest1_LPOS_ToF02_S1W12	Precision Reference	Study Pool	100	TRUE
Sample 3	UnitTest1_LPOS_ToF02_S1W09_x	Assay	Study Sample	100	FALSE
Blank 1	UnitTest1_LPOS_ToF02_Blank01	Assay	Procedural Blank	0	TRUE

Any additional columns in the basic csv file will be appended to the `sampleMetadata` table as additional sample metadata.

### Important Note for LC-MS Datasets

For full nPYc-Toolbox functionality for LC-MS data, there are three additional columns which should be specified, these are:

- ‘Acquired Time’: time of sample acquisition (date/time format)
- ‘Run Order’: the order in which the samples were acquired (an integer value from 1 to the number of samples in ‘Acquisition Time’ order)
- ‘Correction Batch’: the *Analytical Batch* in which each sample was acquired (integer value)

‘Acquired Time’ would routinely be extracted from the raw data (see *Analytical Parameter Extraction* below) and ‘Run Order’ subsequently automatically inferred from this, however, in cases where this is not possible they can be added manually as columns to the ‘Basic CSV’ file.

Similarly, ‘Correction Batch’ can be defined in the ‘Basic CSV’ file, or, if all samples were acquired in the same batch, a column can be added to the sampleMetadata attribute of the LC-MS dataset object after importing into the pipeline by running:

```
msData.sampleMetadata['Correction Batch'] = 1
```

While inclusion of ‘Run Order’ and ‘Correction Batch’ is critical for functionality (namely *Batch & Run-Order Correction* and *Multivariate Analysis*), inclusion of ‘Acquired Time’ does not affect functionality but does enable key plots relying on this data to be generated.

For a full example csv file with these columns included see ‘DEVSET U RPOS Basic CSV.csv’ (*Installation and Tutorials*), but in brief, if adding manually into the ‘Basic CSV’ file the structure of the extra columns might look something like this:

Table 2: Additional columns required for full functionality with LC-MS datasets (note these can be added)

Sample ID	Sample File Name	Assay-Role	Sample-Type	Dilution	Include Sample	Acquired Time	Run Order	Correction Batch
Dilution 1	UnitTest1_LPOS_ToF02_LibA_SRP01	Library Reference	Study Pool	1	TRUE	18/01/2018 02:25:00	1	1
Dilution 2	UnitTest1_LPOS_ToF02_LibA_SRP02	Library Reference	Study Pool	50	TRUE	18/01/2018 02:40:00	2	1
Sample 1	UnitTest1_LPOS_ToF02_AshW07	Assay	Study Sample	100	TRUE	18/01/2018 02:55:00	3	1
Sample 2	UnitTest1_LPOS_ToF02_AshW08	Assay	Study Sample	100	TRUE	18/01/2018 03:10:00	4	1
LTR	UnitTest1_LPOS_ToF02_PSW01	Procession Reference	External Reference	100	TRUE	18/01/2018 03:25:00	5	1
SR	UnitTest1_LPOS_ToF02_PSW02	Procession Reference	SR Study Pool	100	TRUE	18/01/2018 03:40:00	6	1
Sample 3	UnitTest1_LPOS_ToF02_AshW09_x	Assay	Study Sample	100	FALSE	18/01/2018 03:56:00	7	1
Blank 1	UnitTest1_LPOS_ToF02_AshW01	Assay	Procedural Blank	0	TRUE	18/01/2018 04:11:00	8	1

## 6.2 Analytical Parameter Extraction

With the nPYc-Toolbox it is also possible to extract parameters directly from raw data files (currently for Bruker and Waters .RAW data only) and match to the imported dataset using:

```
dataset.addSampleInfo(descriptionFormat='Raw Data', filePath='path to raw data')
```

This links to the underlying `extractParams()` method.

`extractParams` contains several utility functions to read analytical parameters from raw data files.

`nPYc.utilities.extractParams.extractParams(filepath, filetype, pdata=1, whichFiles=None)`

Extract analytical parameters from raw data files for Bruker, Waters .RAW data and .mzML only. :param filepath: Look for data in all the directories under this location. :type searchDirectory: string :param filetype: Search for this type of data :type filetype: string :param int pdata: pdata folder for Bruker data :param list whichFiles: If a list of files is provided, only the files in it will be parsed :return: Analytical parameters, indexed by file name. :rtype: pandas.DataFrame



---

## Sample and Feature Masks

---

Dataset objects contain two internal *mask* vectors, the *sampleMask* and the *featureMask*. They store whether a sample or feature, respectively, should be used when calculating QC metrics, in the visualisations in the report functions and when exporting the dataset.

There are several functions that modify these internal masks:

- *updateMasks()* is a method to automatically mask certain samples and/or features, for example, by type or based on dataset specific quality control checks
- *excludeSamples()* and *excludeFeatures()* are methods to directly exclude specific samples or features respectively. Masked samples and features will remain in the dataset, but will be hidden, and thus ignored when calling the reporting functions, fitting PCA models, and exporting the pre-processed datasets.
- *initialiseMasks()* is a method to reset the masks to include all samples/features.
- *applyMasks()* is a method to permanently exclude from the dataset all samples and features which have been previously masked. After calling this command the excluded features are deleted and the masks are re-initialised so that all remaining samples and features are unmasked. This method should be used only when it is absolutely certain that the masked features and samples are to be removed, as it is permanent so if samples/features were again required, the full dataset would have to be re-imported.

Further details of each of these methods are given below, and full worked examples of how these are used during the import and preprocessing of specific datasets are provided in *Tutorials*.

In brief, however, the following describes a step-wise example utilising the above functions to generate a feature filtered LC-MS dataset containing only *Study Samples*, with a specific sample ('PipelineTesting\_RPOS\_ToF10\_U1W04') excluded:

```
# To automatically mask features not passing quality control criteria
msData.updateMasks(filterFeatures=True, filterSamples=False)

# To reset (initialise) the masks, and run with an updated RSD threshold
msData.initialiseMasks()
dataset.Attributes['rsdThreshold'] = 20
msData.updateMasks(filterFeatures=True, filterSamples=False)
```

(continues on next page)

(continued from previous page)

```
# To automatically mask all sample types except for study samples
msData.updateMasks(filterSamples=True, sampleTypes=[SampleType.StudySample],
↳ assayRoles=[AssayRole.Assay], filterFeatures=False)

# To exclude a specific sample, 'PipelineTesting_RPOS_ToF10_U1W04', by 'Sample File_
↳ Name'
msData.excludeSamples(['PipelineTesting_RPOS_ToF10_U1W04'], on='Sample File Name',
↳ message='User excluded')

# To finally apply the masks and permanently exclude the masked samples and features
msData.applyMasks()
```

## 7.1 Using updateMasks

`updateMasks()` is a method to automatically mask certain samples and/or features, for example, by type or based on dataset specific quality control checks. There are a number of different ways in which the ‘updateMasks’ function can be used, some of which are dataset type specific, common usage includes:

### Using updateMasks to mask samples based on sample type (all dataset types)

By default, all samples are included in the datasets. However, it is often the case that some sample types (see *Recommended Study Design Elements*) would not be required in the final dataset, or when running various quality control checks.

By setting preferences with the “sampleTypes” and “assayRoles” arguments, samples which are not required can also be masked (see :doc:enumerations for possible options). For example, the dataset would be masked to contain only study samples (‘SampleType.StudySample, AssayRole.Assay’) and study reference samples (‘SampleType.StudyPool, AssayRole.PrecisionReference’) by running the following:

```
dataset.updateMasks(filterSamples=True, sampleTypes=[SampleType.StudySample,
↳ SampleType.StudyPool], assayRoles=[AssayRole.Assay, AssayRole.PrecisionReference],
↳ filterFeatures=False)
```

### Using updateMasks to mask samples failing quality control checks (NMR datasets only)

For NMR datasets, there are a number of quality control criteria (Dona *et al*<sup>1</sup>) which are automatically checked (see *Feature Summary Report: NMR Datasets* for full details):

- Chemical shift calibration
- Line width
- Baseline consistency
- Quality of solvent suppression

For each of the above, default acceptable values are given in *Built-in Configuration SOPs*, samples not meeting these criteria can be automatically masked by applying “sampleQCChecks” when applying updateMasks, for example:

```
# To mask all samples failing on any quality control parameter, "sampleQCChecks"
↳ would be set to:
nmrData.updateMasks(filterSamples=True, sampleQCChecks=['CalibrationFail',
↳ 'LineWidthFail', 'BaselineFail', 'SolventPeakFail'], filterFeatures=False)
```

(continues on next page)

<sup>1</sup> Anthony C Dona, Beatriz Jiménez, Hartmut Schäfer, Eberhard Humpfer, Manfred Spraul, Matthew R Lewis, Jake TM Pearce, Elaine Holmes, John C Lindon and Jeremy K Nicholson. Precision High-Throughput Proton NMR Spectroscopy of Human Urine, Serum, and Plasma for Large-Scale Metabolic Phenotyping. *Analytical Chemistry*, 86(19):9887-9894, 2014. URL: <http://dx.doi.org/10.1021/ac5025039>

(continued from previous page)

```
# If only samples failing on Line Width criteria were required to be masked,
→ "sampleQCChecks" would be set to:
nmrData.updateMasks(filterSamples=True, sampleQCChecks=['LineWidthFail'],
→ filterFeatures=False)
```

### Using updateMasks to mask feature failing quality control checks (MS datasets only)

For LC-MS datasets, features should be filtered based on their individual precision and accuracy (Lewis *et al*<sup>2</sup>) in the nPYc-Toolbox the default parameters for feature filtering are as follows:

Table 1: LC-MS Feature Filtering Criteria

Criteria	In	Default Value	Assesses
Correlation to dilution	<i>Serial Dilution Sample</i>	> 0.7	Intensity responds to changes in abundance (accuracy)
<i>Relative Standard Deviation</i> (RSD)	<i>Study Reference</i>	< 30	Analytical stability (precision)
RSD in SS * <i>default value</i> > RSD in SR	<i>Study Sample</i> and <i>Study Reference</i>	1.1	Variation in SS should always be greater than variation in SR

The distribution of correlation to dilution, and RSD can be visualised in the *Feature Summary Report* (see [Quality Assessment Reports](#) for more details).

A report summarising number of features passing selection with different criteria can also be produced using:

```
nPYc.reports.generateReport(dataset, 'feature selection')
```

This generates a list of the number of features passing each filtering criteria, alongside a heatmap showing the number of features resulting from applying different RSD and correlation to dilution thresholds.

Fig. 1: Heatmap of the number of features passing selection with different Residual Standard Deviation (RSD) and correlation to dilution thresholds

Criteria can be modified if required, for example for the RSD threshold using:

```
dataset.Attributes['rsdThreshold'] = 20
```

Features failing selection can be automatically flagged for removal using:

```
dataset.updateMasks(filterSamples=False, filterFeatures=True)
```

### Using updateMasks to mask unwanted/uninformative features (NMR datasets only)

For NMR datasets, feature filtering typically takes the form of removing one or more sections of the spectra known to contain unwanted or un-informative signals.

The regions typically removed are pre-defined in the *Configuration Files*, and can be automatically flagged for removal:

<sup>2</sup> Matthew R Lewis, Jake TM Pearce, Konstantina Spagou, Martin Green, Anthony C Dona, Ada HY Yuen, Mark David, David J Berry, Katie Chappell, Verena Horneffer-van der Sluis, Rachel Shaw, Simon Lovestone, Paul Elliott, John Shockcor, John C Lindon, Olivier Cloarec, Zoltan Takats, Elaine Holmes and Jeremy K Nicholson. Development and Application of Ultra-Performance Liquid Chromatography-TOF MS for Precision Large Scale Urinary Metabolic Phenotyping. *Analytical Chemistry*, 88(18):9004-9013, 2016. URL: <http://dx.doi.org/10.1021/acs.analchem.6b01481>

```
nmrData.updateMasks(filterSamples=False, filterFeatures=True)
```

Additional regions can also be masked by using ‘updateMasks’ with the additional “exclusionRegions” parameter. For example, to also mask the region between 8.4 and 8.5 ppm the following would be run:

```
nmrData.updateMasks(filterSamples=False, filterFeatures=True, exclusionRegions=[(8.4, 8.5)])
```

## 7.2 Using excludeSamples and excludeFeatures

The ‘updateMasks’ function works to mask samples or features not meeting specific criteria, in addition to this, the nPYc-Toolbox also contains two additional methods to mask specific samples or features directly, *excludeSamples()* and *excludeFeatures()* respectively.

These functions both take three input arguments:

1. A list of sample or feature identifiers
2. “on”: the name of the column in ‘sampleMetadata’ (for ‘excludeSamples’) or ‘featureMetadata’ (for ‘excludeFeatures’) where these identifiers can be found
3. “message”: an optional message as to why these samples or features have been flagged for masking

Depending on the dataset type, and the sample and feature metadata available, the value of “on” could differ, but some examples include:

```
# To exclude a sample with 'Sample File Name' = 'DEVSET U 1D NMR raw data files/930'
nmrData.excludeSamples(['DEVSET U 1D NMR raw data files/930'], on='Sample File Name',
    message='Unknown type')

# To exclude all features with 'ppm' > 8
nmrData.excludeFeatures([nmrData.featureMetadata['ppm'][nmrData.featureMetadata['ppm']
    > 8].values], on='ppm', message='ppm > 8')

# To exclude a sample with 'Run Order' = 93:
msDatacorrected.excludeSamples([93], on='Run Order', message='outlying TIC')
```

## 7.3 Using applyMasks and initialiseMasks

Once satisfied with the sample and feature masks, exclusions can be applied (permanently removed from the dataset) using the *applyMasks()* function:

```
msDatacorrected.applyMasks()
```

This method should be used only when it is absolutely certain that the masked features and samples are to be removed, as the full dataset would otherwise have to be re-imported.

Before masks have been applied, however, feature/sample masking can be reset to include all samples/features using *initialiseMasks()*:

```
msDatacorrected.initialiseMasks()
```



---

## Quality Assessment Reports

---

The nPYc-Toolbox offers a series of reports, pre-set visualisations comprised of text, figures and tables to describe and summarise the characteristics of the dataset, and help the user assess the overall impact of quality control decisions (e.g. whether to exclude samples or features or change filtering criteria).

The main reporting functions include:

- **Sample Summary:** Presents a summary of the samples acquired, see below for details
- **Feature Summary:** Summarises the main properties of the dataset and method specific quality control metrics, see below for details
- **Multivariate Report:** Summarises the main outputs of a PCA model and any potential associations with pertinent analytical metadata, see [Multivariate Analysis](#) for full details
- **Final Report:** Summary report compiling information about the samples acquired, and the overall quality of the dataset
- **Batch and Run-Order Correction:** Specific reports for optimising and assessing correction in *MSDataset*, see [Batch & Run-Order Correction](#) for full details
- **Feature Selection:** specific report for assessing the number of features passing quality criteria in *MSDataset*, see [Sample and Feature Masks](#) for full details

By default, reports are generated inline (i.e. in a Jupyter notebook), using `generateReport()`. However reports can also be saved as html documents with static images by supplying a destination path, for example:

```
saveDir = '/path to save outputs'
nPYc.reports.generateReport(dataset, 'feature summary', destinationPath=saveDir)
```

The html versions of the reports use Jinja2 templates, default reports are saved in the *Templates* directory, and may be customised if required.

By default, reports are generated on the full sample and feature complement of each dataset, however, reports can also be generated on only those samples and features not set to be masked from the dataset (i.e. with `sampleMask` and `featureMask` values set to `True`, see [Sample and Feature Masks](#)), by running with `withExclusions=True` for example:

```
nPYc.reports.generateReport(dataset, 'feature summary', withExclusions=True)
```

In this way, samples and features can be iteratively masked/included, and the impact of masking visualised before the masks are finally applied and samples and/or features permanently excluded from the dataset.

Throughout the reports, we reference the various QC sample types included in every dataset to enable the characterisation of data quality, see [Sample Metadata](#) for full details.

## 8.1 Sample Summary Report

The sample summary report can be used to check the expected samples against those acquired, in terms of numbers, sample type, and any samples either missing from acquisition or not recorded in the sample metadata CSV file:

```
nPYc.reports.generateReport(msData, 'sample summary')
```

The main underlying function parameters are as follows:

**class** nPYc.reports.\_generateSampleReport

Summarise samples in the dataset.

Generate sample summary report, lists samples acquired, plus if possible, those missing as based on the expected sample manifest.

### Parameters

- **dataTrue** (*Dataset*) – Dataset to report on
- **withExclusions** (*bool*) – If True, only report on features and samples not masked by the sample and feature masks
- **destinationPath** (*None or str*) – If None, run interactively, else a str specifying the directory to save report into
- **returnOutput** (*bool*) – If True, returns a dictionary of all tables generated during run

**Returns** Optional, dictionary of all tables generated during run

## 8.2 Feature Summary Report: LC-MS Datasets

The LC-MS feature summary report provides visualisations summarising the quality of the dataset with regards to quality control criteria previously described in Lewis *et al*<sup>1</sup> and can be run using:

```
nPYc.reports.generateReport(msData, 'feature summary')
```

The visualisations include both assessment of potential run-order and batch effects, and metrics by which feature quality can be assessed, in order, these consist of:

- Feature abundance (Figure 1)
- Sum of total ion count, TIC (Figures 2 and 3)

<sup>1</sup> Matthew R Lewis, Jake TM Pearce, Konstantina Spagou, Martin Green, Anthony C Dona, Ada HY Yuen, Mark David, David J Berry, Katie Chappell, Verena Horneffer-van der Sluis, Rachel Shaw, Simon Lovestone, Paul Elliott, John Shockcor, John C Lindon, Olivier Cloarec, Zoltan Takats, Elaine Holmes and Jeremy K Nicholson. Development and Application of Ultra-Performance Liquid Chromatography-TOF MS for Precision Large Scale Urinary Metabolic Phenotyping. *Analytical Chemistry*, 88(18):9004-9013, 2016. URL: <http://dx.doi.org/10.1021/acs.analchem.6b01481>

- Correlation to dilution (Figures 4, 5 and 7)
- Residual standard deviation, RSD (Figures 6, 7 and 9)
- Chromatographic peak width, if available (Figure 8)
- Ion map (Figure 10)

For several of these parameters (for example, correlation to dilution, RSD), acceptable default values are pre-defined in the configuration SOP, see [Built-in Configuration SOPs](#) for details. If different values are required, these can be set by the user either by modifying the SOP, or during data import, or directly at any point during running the pipeline. For more information, see [Datasets](#) and for examples, [Installation and Tutorials](#).

The following sections describe how the quality for each of these is assessed.

### Feature abundance

The histogram of feature abundance shows the distribution of mean abundance by sample type for each feature (Figure 1).

Fig. 1: Figure 1: Feature intensity histogram for all samples and all features in dataset (by sample type).

While a normal distribution is expected for the SS and SR samples, if your study includes LTR samples (QC samples from a different source to the study) it can be the case that a subset of features are not present in these samples. If this is the case, and it is required to limit the feature set to those detected in your LTR samples, features not found in this set could be excluded based on their intensity (see [Sample and Feature Masks](#) for details). If an unexpected distribution is observed this should be investigated, for example, by going back to XCMS feature extraction parameters.

### Sum of total ion count, TIC

The TIC plot shows the summed intensity of all feature integrals for each sample (Figure 2) and provides insight into potential run-order and batch effects.

Fig. 2: Figure 2: Sample Total Ion Count (TIC) and distribution (coloured by sample type).

By plotting the TIC for each sample (ordered by acquisition date) any broad trends in overall sample intensity can be observed. With LC-MS it is usual to see a gradual decline in TIC across the run owing to increasing inefficiencies in ion detection (from source and ion optic contamination), alongside large jumps if data is acquired in multiple batches, both of which can be mitigated (at least in part) by run-order and batch correction (see [Batch & Run-Order Correction](#)).

With our instrumental set-up (recent generation Waters QToF instruments), we implement an automatic gain control (see Lewis *et al*<sup>1</sup>). Briefly, throughout each experiment, the voltage applied to the MS detector is automatically adjusted to compensate for trends in instrument performance, which, especially when the increments in applied voltage are large, has a noticeable effect on the total ion count (TIC) of the sample. Although with our current set-up changes in detector voltage are capped and thus this is minimised, this was not always (and may not always be) the case. Therefore, an additional figure of TIC coloured by detector voltage is provided (see Figure 3 in tutorial).

### Correlation to dilution

Correlation to dilution is one metric by which feature quality can be determined. By inclusion of a dilution series ([Serial Dilution Sample](#), SRD) the correlation to dilution for each feature can be calculated. A histogram of the resulting values shows the distribution of correlation to dilution (Figure 4) and a TIC plot for the SRD samples can be used to assess the overall behaviour of the dilution series (Figure 5).

A high quality dataset should contain only features that can be shown to be measured accurately with respect to the true intensity, i.e. to scale with dilution. During feature filtering, a threshold in correlation to dilution (default value 0.7) is used to exclude all features which do not respond to dilution (see [Sample and Feature Masks](#) for details). Figure 4 shows the distribution in correlation to dilution segmented by mean feature intensity. If the distribution in correlation

to dilution values is not highly skewed to high values (especially for high and medium intensity features), the reason for this needs investigating.

Fig. 3: Figure 4: Histogram of pearson correlation of features to serial dilution, segmented by percentile.

The first thing to check in this case is that the overall trend in TIC for the dilution series samples corresponds to the expected dilution as defined in the 'Basic CSV' file (see *CSV template for metadata import*), this is shown in Figure 5. Any outliers (for example, mis-injections) can be excluded, which may have a substantial impact on the resulting correlation values.

Fig. 4: Figure 5: TIC of serial dilution (SRD) samples coloured by sample dilution.

If a large number of SRD samples are not scaling with dilution, and the distribution in correlation values is poor, the cause of this should be investigated across all stages, from acquisition, through conversion and peak detection.

### Residual standard deviation, RSD

Another key metric by which feature quality can be assessed is that of residual standard deviation (*Relative Standard Deviation*, RSD). By inclusion of precision reference samples (*Study Reference*, SR) or *Long-Term Reference*, LTR) the RSD for each feature can be calculated. A histogram of the resulting values shows the distribution of RSD in the SR samples (Figure 6) and a plot of the RSD for each feature by sample type (Figure 9) allows comparison of the variation observed between QC and study samples.

A high quality dataset should contain only features that can be shown to be measured precisely from multiple acquisitions across the run (in this case this is provided by repeated injections of the pooled SR sample). During feature filtering a threshold in RSD (default value 30) is used to exclude all features which cannot be measured precisely across the run (see *Sample and Feature Masks* for details). Figure 6 shows the distribution in RSD segmented by mean feature intensity. If the distribution is not skewed to low values (especially for high and medium intensity features), the reason for this needs investigating.

Fig. 5: Figure 6: Histogram of Residual Standard Deviation (RSD) in study reference (SR) samples, segmented by abundance percentiles.

The first thing to check is substantial run-order and batch trends (Figure 2), if these are present, the RSD in the SR samples will be skewed to higher values, and batch and run-order correction should be first applied. Additionally, outlying SR samples can cause inflation to the RSD, if a small number of SR samples demonstrate an unusual TIC (which is not shown by surrounding SS samples) these should be excluded before RSD is calculated.

In addition to the requirement that features are measured precisely, the variance observed in the study samples, should exceed that measured in the SR samples, with the expectation that biological variance should exceed analytical variance. The plot comparing the RSD measured in the different sample classes (study reference sample, study samples etc.) provides insight into variance structures in the dataset (Figure 9).

Finally, to assess the main feature quality metrics together a plot of RSD vs. correlation is provided (see Figure 7 in tutorial).

### Chromatographic peak width

If available, a histogram is plotted of chromatographic peak width (if available, Figure 8).

Narrower peaks mean better chromatographic resolution, while broadening in peak width (when compared with previous runs) imply indicate potential aging of the column, which may need replacing.

### Ion map

The ion map visualises the location of the detected features in the  $m/z$  and retention time space of the assay (Figure 10).

Fig. 6: Figure 9: RSD distribution for all samples and all features in dataset (by sample type).

Fig. 7: Figure 10: Ion map of all features (coloured by log median intensity).

This plot can be used to assess potential feature exclusion ranges. For example, where the retention time is outside the useful range of the assay, or presence of signals resulting from polymer contamination.

## 8.3 Feature Summary Report: NMR Datasets

The NMR feature summary report provides visualisations summarising the quality of the dataset with regards to quality control criteria previously described in Dona *et al*<sup>2</sup> and can be run using:

```
nPYc.reports.generateReport(nmrData, 'feature summary')
```

The visualisations include various metrics by which dataset quality can be assessed, in order, these consist of:

- Chemical shift calibration (Figure 1)
- Line width (Figures 2 and 3)
- Baseline consistency (Figure 4)
- Quality of solvent suppression (Figure 5)

For several of these parameters (for example, line width), acceptable default values are pre-defined in the configuration SOP, see *Built-in Configuration SOPs* for details. If different values are required, these can be set by the user either by modifying the SOP, or during data import, or directly at any point during running the pipeline. For more information, see *Datasets* and for examples, *Installation and Tutorials*.

Any samples failing any of the above criteria are flagged, but in the appropriate plots, and in the table at the end of the report.

The following sections describe how the quality for each of these is assessed.

### Chemical shift calibration

Variations in sample temperature between acquisitions can result in minor deviations in the chemical shift scale between spectra. To correct these shifts, the toolbox uses an adaptation of the technique published in Pearce *et al*<sup>3</sup>.

Subsequently, the chemical shift calibration algorithm detects deviation from the expected delta ppm and flags those samples outside of the empirical 95% bound as estimated from the whole dataset (Figure 1).

Fig. 8: Figure 1: Distribution of all samples after chemical shift calibration (5-95% with outliers flagged).

If spectra are failing calibration, firstly the presence of the target resonance should be checked, and if required, a different peak target can be defined as described above.

### Line width

<sup>2</sup> Anthony C Dona, Beatriz Jiménez, Hartmut Schäfer, Eberhard Humpfer, Manfred Spraul, Matthew R Lewis, Jake TM Pearce, Elaine Holmes, John C Lindon and Jeremy K Nicholson. Precision High-Throughput Proton NMR Spectroscopy of Human Urine, Serum, and Plasma for Large-Scale Metabolic Phenotyping. *Analytical Chemistry*, 86(19):9887-9894, 2014. URL: <http://dx.doi.org/10.1021/ac5025039>

<sup>3</sup> Jake TM Pearce, Toby J Athersuch, Timothy MD Ebbels, John C Lindon, Jeremy K Nicholson and Hector C Keun. Robust Algorithms for Automated Chemical Shift Calibration of 1D 1H NMR Spectra of Blood Serum. *Analytical Chemistry*, 80(18):7158-62, 2008. URL: <http://dx.doi.org/10.1021/ac8011494>

Spectral line-width is calculated by fitting a pseudo-voigt line shape to a pre-specified signal on the native-resolution Fourier-transformed spectrum at import, using the *lmfit* module <sup>(4)</sup> to optimise the fit.

In the default configuration of the toolbox, line-width is calculated by fitting the TSP singlet at (ppm=0) in urine spectra, and the lactate quartet at (ppm=4.11) in serum or plasma.

A box plot of the calculated line width values coloured by sample type is plotted in Figure 2.

Fig. 9: Figure 2: Boxplot of line width values (coloured by sample type).

Any samples with values above the line width threshold (here set to 0.8 as above), are also plotted (Figure 3)

Fig. 10: Figure 3: Distribution of all samples around peak on which line width calculated (5-95% with outliers flagged).

Depending on the number of samples failing the line width checks, either individual samples may be re-run, or, if necessary, the acquisition parameters adjusted by the spectroscopist.

### Baseline consistency

Baseline consistency is calculated based on two regions at either end of the spectrum expected to contain only electronic noise. For these regions the 5% and 95% percentile bounds in intensity are calculated using all the points in all the spectra. For each individual spectrum, if more than 95% of the intensity points fall outside of these bounds the sample is flagged for review (Figure 4).

Fig. 11: Figure 4: Distribution of all samples at baseline regions (5-95% with outliers flagged).

The phasing of spectra flagged for review should first be checked, and adjusted if applicable. If a larger number of samples in the dataset fail the spectrometer acquisition parameters (such as receiver gain settings) and sample preparation (such as dilution) should be revised.

### Quality of solvent suppression

The solvent suppression quality control is performed by applying the same method as above to the regions flanking either side of the residual solvent peak (Figure 5).

This test normally flags very dilute samples for which it might be difficult to obtain a high quality spectrum without adjusting the sample preparation. However, for these spectra, re-acquisition with more manual adjustment of the solvent suppression parameters may substantially improve the data.

## 8.4 Feature Summary Report: NMR Targeted Datasets

The feature summary report provides visualisations summarising the quality and distribution of values across samples for each individual feature. This report can be obtained by running:

```
nPYc.reports.generateReport(TargetedData, 'feature summary')
```

In order, for an NMR targeted dataset these consist of:

- Summary of quantification parameters (Table 1)

---

<sup>4</sup> Matt Newville, Renee Otten, Andrew Nelson, Antonino Ingargiola, Till Stensitzki, Dan Allan, Austin Fox, Michał, Glenn, Yoav Ram, MerlinSmiles, Li Li, Christoph Deil, Stuermer, Alexandre Beelen, Oliver Frost, gpasquev, Allan L. R. Hansen, Alexander Stark, Tim Spillane, Shane Caldwell, Anthony Polloreno, Nicholas Earl, colgan, Robbie Clarken, Kostis Anagnostopoulos, Jose Borreguero, deep-42-thought, Ben Gamari and Anthony Almarza. lmfit. 2018. URL: <https://doi.org/10.5281/zenodo.1249416>

Fig. 12: Figure 5: Distribution of all samples at region surrounding solvent suppression peak (5-95% with outliers flagged).

- Residual standard deviation, RSD (Figure 2, Table 2)
- Feature distributions (Figure 3)

### Summary of quantification parameters

The initial set of tables in the targeted feature summary report summarise information about each of the quantified features (including quantification parameters and reference ranges if available). The first table gives overall results, and in subsequent tables the features and results are broken down by the quantification type.

Depending on the data generation process, the confidence in quantified values can vary greatly, ranging from semi-quantitative measurements (where area is reported but not concentration) to quantitative values (where absolute concentrations are reported) (Broadhurst *et al*<sup>5</sup>).

The *QuantificationType* field describes the rigour of the quantification of each compound, see *class nPYc.enumerations.QuantificationType* for all available options. In Bruker NMR Targeted methods, compounds are quantified using a quantitative method that does not rely on internal standards, therefore ‘QuantOther’ is the recorded value.

Similarly, as a variety of calibration methods can be employed, the *CalibrationMethod* defines how the calibration curve and spiked standards interact to establish a quantitative measurement, see *class nPYc.enumerations.CalibrationMethod* for all available options. In Bruker NMR Targeted methods, a quantitative approach that does not rely on calibration curves and internal standard is utilised, therefore ‘otherCalibration’ is the recorded value.

For a given analytical platform, each compound will have a range at which its concentration can be satisfactorily determined; outside of which range the reported value could substantially differ from the true sample concentration (Synovec *et al*<sup>6</sup>). The definition of what is “satisfactory” and how this range (sometimes called linear range) is determined is specific to the analytical platform and common guidelines set by the community and regulatory agencies (Lee *et al*<sup>7</sup>).

Depending on the quantification method employed, there are several different quantification measures that may be reported. The LLOQ (lowest limit of quantification) and ULOQ (upper limit of quantification) are the lowest and highest concentration values respectively between which quantitative results can be obtained with a specific degree of confidence. When reporting quantitative data, current convention impose to report values inferior to the LLOQ as “<LLOQ” and values superior to the ULOQ as “>ULOQ”. Data extrapolated outside of these limits are typically not reported in published results as they do not satisfy a predefined degree of confidence

Alternatively, limits of detection (LOD) are sometimes reported, as is the case for the Bruker NMR Targeted outputs.

### Residual standard deviation, RSD

As for the MS profiling datasets, for targeted datasets *Relative Standard Deviation* can be calculated for each feature, and on each sample type (Figure 2, Table 2).

Fig. 13: Figure 2: Distribution of values for each feature per sample type.

<sup>5</sup> David Broadhurst, Royston Goodacre, Stacey N Reinke, Julia Kuligowski, Ian D Wilson, Matthew R Lewis and Warwick B Dunn. Guidelines and considerations for the use of system suitability and quality control samples in mass spectrometry assays applied in untargeted clinical metabolomic studies. *Metabolomics*, 14(6):72, 2018. URL: <https://doi.org/10.1007/s11306-018-1367-3>

<sup>6</sup> Synovec, Robert E and Yeung, Edward S. Improvement of the Limit of Detection in Chromatography by an Integration Method. *Analytical Chemistry*, 57(12):2162-2167, 1985. URL: <https://doi.org/10.1021/ac00289a001>

<sup>7</sup> Jean W Lee, Viswanath Devanarayan, Yu Chen Barrett, Russell Weiner, John Allinson, Scott Fountain, Stephen Keller, Ira Weinryb, Marie Green, Larry Duan, James A Rogers, Robert Millham, Peter J O’Brien, Jeff Sailstad, Masood Khan, Chad Ray and John A Wagner. Fit-for-purpose method development and validation for successful biomarker measurement. *Pharmaceutical Research*, 23(2):312-28, 2006. URL: <http://dx.doi.org/10.1007/s11095-005-9045-3>



Figure 2 allows a comparative visualization of the RSD per feature across each Sample Type.

A high quality dataset should contain only features that can be shown to be measured precisely from multiple acquisitions across the run (in this case this is provided by repeated injections of the pooled SR sample). Subsequently, at the feature filtering stage a threshold in RSD (default value 30) is used to exclude all features that cannot be measured precisely across the run.

From both Figure 2 and Table 2 it can be seen that for this dataset there are many features with zero values across all samples (and thus also an RSD of zero), these features can also be removed from the dataset if required.

### Feature distributions

Finally, violin plots giving the distribution in intensity for each measured feature and for each sample type are shown in Figure 3.

Fig. 14: Figure 3: Distribution of values for each feature per sample type.

These can also be used to identify features with a very high proportion of zeros or values outside the limits of quantification.

## 8.5 Dataset Specific Reporting Syntax and Parameters

The main function parameters (which may be of interest to advanced users) are as follows:

**class** nPYc.reports.\_generateReportMS

Summarise different aspects of an MS dataset

Generate reports for feature summary, correlation to dilution, batch correction assessment, batch correction summary, feature selection, final report, final report abridged, or final report targeted abridged

- **‘feature summary’** Generates feature summary report, plots figures including those for feature abundance, sample TIC and acquisition structure, correlation to dilution, RSD and an ion map.
- **‘correlation to dilution’** Generates a more detailed report on correlation to dilution, broken down by batch subset with TIC, detector voltage, a summary, and heatmap indicating potential saturation or other issues.
- **‘batch correction assessment’** Generates a report before batch correction showing TIC overall and intensity and batch correction fit for a subset of features, to aid specification of batch start and end points.
- **‘batch correction summary’** Generates a report post batch correction with pertinent figures (TIC, RSD etc.) before and after.
- **‘feature selection’** Generates a summary of the number of features passing feature selection (with current settings as definite in the SOP), and a heatmap showing how this number would be affected by changes to RSD and correlation to dilution thresholds.
- **‘final report’** Generates a summary of the final dataset, lists sample numbers present, a selection of figures summarising dataset quality, and a final list of samples missing from acquisition.
- **‘final report abridged’** Generates an abridged summary of the final dataset, lists sample numbers present, a selection of figures summarising dataset quality, and a final list of samples missing from acquisition.
- **‘final report targeted abridged’** Generates an abridged summary of the final targeted (peakPantheR) dataset, lists sample numbers present, a selection of figures summarising dataset quality, feature distributions, and a final list of samples missing from acquisition.



### Parameters

- **msDataTrue** (*MSDataset*) – MSDataset to report on
- **reportType** (*str*) – Type of report to generate, one of feature summary, correlation to dilution, batch correction, feature selection, final report, final report abridged, or final report targeted abridged
- **withExclusions** (*bool*) – If True, only report on features and samples not masked by the sample and feature masks
- **or bool withArtifactualFiltering** (*None*) – If None use the value from `Attributes['artifactualFilter']`. If True apply artifactual filtering to the feature selection report and final report
- **destinationPath** (*None or str*) – If None plot interactively, otherwise save report to the path specified
- **msDataCorrected** (*MSDataset*) – Only if batch correction, if msDataCorrected included will generate report post correction
- **pcaModel** (*PCAmode1*) – Only if final report, if PCAmodel object is available PCA scores plots coloured by sample type will be added to report

**class** nPYc.reports.\_generateReportNMR

Generate reports on NMRdataset objects, possible options are: feature summary or final report

- **‘feature summary’** Generates feature summary report/ QC summary report, plots figures including those for feature calibration check against glucose or TSP, linewidth box plot and baseline/water peak plots.
- **‘final report’** Generates a summary of the final dataset, lists sample numbers present, a selection of figures summarising dataset quality, and a final list of samples missing from acquisition.

### Parameters

- **nmrData** (*NMRDataset*) – NMRDataset to report on
- **reportType** (*str*) – Type of report to generate, one of feature summary, or final report
- **withExclusions** (*bool*) – If True, only report on features and samples not masked by the sample and feature masks
- **destinationPath** (*None or str*) – If None plot interactively, otherwise save report to the path specified

**class** nPYc.reports.\_generateReportTargeted

Summarise different aspects of a Targeted Dataset

Generate reports for feature summary, merge LOQ assessment or final report

- **‘feature summary’** Generates feature summary report, ...
- **‘merge loq assessment’** Generates a report before `mergeLimitsOfQuantification()`, highlighting the impact of updating limits of quantification across batch. List and plot limits of quantification that are altered, number of samples impacted.
- **‘final report’** Generates a summary of the final dataset, lists sample numbers present, a selection of figures summarising dataset quality, and a final list of samples missing from acquisition.

### Parameters

- **tDataIn** (*TargetedDataset*) – TargetedDataset to report on
- **reportType** (*str*) – Type or report to generate, one of feature summary, merge loq assessment or final report
- **withExclusions** (*bool*) – If True, only report on features and samples not masked by sample and feature masks
- **destinationPath** (*None or str*) – If None plot interactively, otherwise save report to the path specified
- **numberPlotPerRowLOQ** (*int*) – Only if merge loq assessment, the number of subplots to place on each row
- **numberPlotPerRowFeature** (*int*) – Only if feature summary or final report, the number of subplots to place on each row
- **percentRange** (*None or float*) – None or Float, percentage range for acceptable accuracy [100 - percentRange, 100 + percentRange] and precision [0, percentRange]
- **pcaModel** (*PCAmodel*) – Only if final report, if PCAmodel object is available PCA scores plots coloured by sample type will be added to report

#### Raises

- **ValueError** – If ‘tData’ does not satisfy to BasicTargetedDataset definition
- **ValueError** – If ‘reportType’ is not feature summary, merge LOQ assessment or final report
- **TypeError** – If ‘withExclusion’ is not a bool
- **TypeError** – If ‘destinationPath’ is not None or str
- **TypeError** – If ‘numberPlotPerRowLOQ’ is not int
- **TypeError** – If ‘numberPlotPerRowFeature’ is not int
- **TypeError** – If ‘percentRange’ is not None or float

---

## Batch & Run-Order Correction

---

The *batchAndROCorrection* module provides tools to detect and correct for per-feature run-order and batch effects in *MSDataset*, by characterising the effect in reference samples and interpolating a correction factor to the intermediate samples.

Fig. 1: TIC (Total Ion Count) plot showing the summed intensity of all feature integrals for each sample (coloured by sample type) before batch and run-order correction.

Run-order and batch correction may be applied following an adapted version of the LOWESS approach proposed by Dunn *et al*<sup>1</sup>.

Fig. 2: TIC (Total Ion Count) plot showing the summed intensity of all feature integrals for each sample (coloured by sample type) after batch and run-order correction.

In brief, for each MS feature, a LOWESS estimator is fitted on the series of consecutive *Study Reference* samples for each analytical batch (which can be defined by the user, see *Sample Metadata*). The value for that feature in each sample is corrected by dividing the original intensity value by the interpolated value of the LOWESS curve at its position in the run order (final intensity units are a ratio to intensity in the "mean" *Study Reference* sample expressed by the LOWESS curve). The window of the LOWESS smoother can be set by the user (default value=11), care should be taken not to over-fit the run-order correction. Batch divergences are corrected by aligning median feature intensities in the *Study Reference* samples between batches.

As batch and run-order correction is a critical step in preprocessing of LC-MS datasets, alongside further information below, a full and detailed example is given in the LC-MS tutorial, see *Installation and Tutorials*.

---

<sup>1</sup> Warwick B Dunn, David Broadhurst, Paul Begley, Eva Zelena, Sue Francis-McIntyre, Nadine Anderson, Marie Brown, Joshau D Knowles, Antony Halsall, John N Haselden, Andrew W Nicholls, Ian D Wilson, Douglas B Kell, Royston Goodacre, and The Human Serum Metabolome (HUSERMET) Consortium. Procedures for large-scale metabolic profiling of serum and plasma using gas chromatography and liquid chromatography coupled to mass spectrometry. Nature Protocols, 6:1060 EP –, 06 2011. URL: <http://dx.doi.org/10.1038/nprot.2011.335>.

## 9.1 Batch & Run-Order Correction Assessment

Batch & run-order correction performance can be assessed on a subset of features prior to running on the whole dataset using the *Batch Correction Assessment* report:

```
nPYc.reports.generateReport(msData, 'batch correction assessment', batch_correction_
↪window=11)
```

This report shows the LOESS fit for a number of features (default 10), and the results of applying such a fit.

Fig. 3: Example outcome of applying batch and run-order correction to one feature, plot shows an arrow for each sample from the summed intensity (TIC) before to after batch correction, plus the LOESS fit.

By comparing the results across all surveyed features, the parameters for and necessity of correction can be assessed:

- Is the window of the LOWESS smoother appropriate? Check that only broad and not narrow trends are being fitted, change `batch_correction_window` parameter if required.
- Does the correction need to be applied in different sub-batches? Check if there is a common and consistent jump in intensity across all features, amend the sample batches if required.
- Do any SR samples need to be excluded? If you have a non-representative consecutive set of SR samples in your dataset, they may need removing.
- Is batch correction required? Check if there is an observable trend in the batch and/or run-order, if not then correction is not required!

Once these questions have been assessed, the appropriate parameters can be modified, or samples excluded, for full details and a worked example see the LC-MS tutorial at *Installation and Tutorials*.

## 9.2 Running Batch & Run-Order Correction

Batch and run-order correction can be applied to a *MSDataset* using:

```
datasetCorrected = nPYc.batchAndROCorrection.correctMSdataset(dataset, window=11)
```

After running correction, the results can be assessed using the *Batch Correction Summary* report:

```
nPYc.reports.generateReport(dataset, 'batch correction summary',
↪msDataCorrected=datasetCorrected)
```

The main function parameters (which may be of interest to advanced users) are as follows:

```
nPYc.batchAndROCorrection.correctMSdataset(data, window=11, method='LOWESS',
align='median', parallelise=True, exclude-
Failures=True)
```

Conduct run-order correction and batch alignment on the *MSDataset* instance *data*, returning a new instance with corrected intensity values.

Sample are separated into batches according to the 'Correction Batch' column in *data.sampleMetadata*.

### Parameters

- **data** (*MSDataset*) – *MSDataset* object with measurements to be corrected
- **window** (*int*) – When calculating trends, consider this many reference samples, centred on the current position

- **method** (*str*) – Correction method, one of ‘LOWESS’ (default), ‘SavitzkyGolay’ or None for no correction
- **align** (*str*) – Average calculation of batch and feature intensity for correction, one of ‘median’ (default) or ‘mean’
- **parallelise** (*bool*) – If True, use multiple cores
- **excludeFailures** (*bool*) – If True, remove features where a correct fit could not be calculated from the dataset

**Returns** Duplicate of *data*, with run-order correction applied

**Return type** *MSDataset*



The nPYc-Toolbox provides the capacity to generate a PCA model of the data (via the pyChemometrics module), and subsequently, to use this to assess data quality, identify potential sample and feature outliers, and determine any potential analytical associations with the main sources of variance in the data.

## 10.1 PCA Model

A PCA model can be generated using `exploratoryAnalysisPCA()`:

```
PCAmoel = nPYc.multivariate.exploratoryAnalysisPCA(dataset, scaling=1)
```

There are a number of parameters which can be optimised depending on the dataset.

One key parameter is ‘scaling’, which divides each column in the data matrix by its respective standard deviation raised to a power of the scaling parameter. This parameter can range in value between 0 and 1, and recommended values are 0 for mean-centering only, 0.5 for Pareto scaling and 1 for unit variance (UV) scaling. The outcome of PCA model will vary based on the scaling method selected, and different scaling functions can be appropriate depending on the data itself and the question being asked of the data (van der Berg *et al*<sup>1</sup>).

The default scaling is unit variance (*scaling=1*), which scales every variable to have a variance of one, and thus all variables (despite their different magnitudes) become equally important in the model. For NMR, when smaller variables are more likely to be background noise, it may be that mean-centering the data only (*scaling=0*) can be appropriate.

Each model is cross-validated using 7-fold cross-validation and the recommended number of principal components is automatically estimated based on two criteria, when either one of these is met no more components will be added and the PCA model will be returned. There criteria are:

- *minQ2*: Q2 is the variance predicted by each component (from cross-validation), when adding a component does not improve Q2 by at least this value (default *minQ2=0.05*) then no more components will be added.

<sup>1</sup> Robert A van den Berg, Huub CJ Hoefsloot, Johan A Westerhuis, Age K Smilde and Mariët J van der Werf. Centering, scaling, and transformations: improving the biological information content of metabolomics data. BMC Genomics, 8(7):142, 2006. URL: <https://doi.org/10.1186/1471-2164-7-142>

- *maxComponents*: this defines the maximum number of components (default *maxComponents=10*) returned by the model (regardless of Q2 increases).

If different parameters are required these can be specified as additional input arguments, for example, to generate a PCA model with a maximum of five components:

```
PCAmodel = nPYc.multivariate.exploratoryAnalysisPCA(dataset, scaling=1, ↵  
↵maxComponents=5)
```

The main function parameters (which may be of interest to advanced users) are as follows:

```
nPYc.multivariate.exploratoryAnalysisPCA.exploratoryAnalysisPCA(npycDataset,  
                                                                scaling=1,  
                                                                maxCompo-  
                                                                nents=10,  
                                                                minQ2=0.05,  
                                                                withExclu-  
                                                                sions=False,  
                                                                **kwargs)
```

Performs and exploratory analysis using PCA on the data contained in an `Dataset`.

#### Parameters

- **npycDataset** (`Dataset`) – Dataset to model
- **scaling** – Choice of scaling.
- **maxComponents** (`int`) – Maximum number of components to fit.
- **minQ2** – Minimum % of improvement in Q2Y over the previous component to add .
- **withExclusions** (`Boolean`) – If True, PCA will be fitted on the `npyc_dataset` after applying feature and sample Mask, if False the PCA is performed on whole dataset.

**Returns** Fitted PCA model

**Return type** ChemometricsPCA

## 10.2 Multivariate Analysis Report

The analytical multivariate report provides visualisations summarising the largest sources of variance in the dataset (from a PCA model) with particular emphasis on any potential analytical sources, and can be generated using `multivariateReport()`:

```
nPYc.reports.multivariateReport(dataset, PCAmodel)
```

These consist of:

- Scree plot of variance (Figure 1)
- Scores plots coloured by sample type (Figure 2)
- Strong sample outliers (Figure 3)
- DmodX sample outliers (Figure 4)
- Loadings plots (Figure 5)
- Distribution of analytical parameters (Figure 6)
- Heatmap of potential associations between analytical parameters and the main sources of variance (Figures 7 and 8)



- Scores plots coloured by analytical parameters with potential association (Figures 9-11)

The following sections describe these in more detail:

### Scree plot of variance

The scree plot (Figure 1) shows the percentage of variance (cumulative) both explained (R2) and predicted (Q2) by each principal component in the model.

Fig. 1: Figure 1. PCA scree plot of variance explained by each component (cumulative).

This information can help to guide interpretation of the subsequent plots, for example, if separation is seen between QC samples in a given component, this would be much more serious if this component explained 50% of the variance than if the component only explained 3% of the variance.

If the variance is not predictable (negative or low Q2) this indicates that the model is not robust to different subsets of samples being removed. This could be for a number of reasons, but it implies that there are likely no key analytical sources of variance, as these would inherently contribute throughout the run.

### Scores plots coloured by sample type

The PCA scores represent the new location of the samples in each principal component. A typical way to look at these is to plot the scores values in two dimensions, corresponding to pairs of components. Each point in a scores plot therefore represents a sample, with samples close together being more similar to each other, and those further apart being more dissimilar. By colouring by sample type (Figure 2), we can check, firstly, the consistency of the QC samples (SR and LTR) i.e. that they are tightly clustered; and secondly, for the presence of any sample outliers (any samples which are very different to the others).

Fig. 2: Figure 2. PCA scores plots coloured by sample type.

Outlying study samples in this plot are of interest (does this relate to biology, sample collection etc.), but not of particular concern. However, any QC samples with unusual behaviour, or unusual clustering within QC samples, should be checked. Further sections of the multivariate report can help to elucidate if this behaviour may result from an analytical source (see below).

### Strong sample outliers

Strong sample outliers are those which are very different from the others, these are seen as far outside the Hotelling's T ellipse (Figure 2) and with high values when you sum the total distance from origin across all components (Figure 3), these samples have high leverage and skew the model with their contribution to variance.

Fig. 3: Figure 3. Distribution in total distance from origin (scores space) by sample type.

Outliers in study samples are common, owing for example, to the presence of strong drug or dietary related signals. If LTR samples are strong outliers this is not a concern, as they may have a significantly different composition to the study samples, similarly SR samples may be overly weighted with very high concentration metabolites in one or more study sample, so may also be located away from the origin.

It would be expected here for all LTR and all SR samples to have roughly the same total distance from origin (although the two groups may be different from each other), as above, any deviation from this should be investigated.

### DmodX sample outliers

DmodX (also called distance to model) is a measure of how well each sample fits the model itself. Unlike the strong sample outliers, outliers in DmodX (Figure 4) do not skew the model, but are simply not well represented by the model.

Fig. 4: Figure 4. Distribution in distance from model (DmodX) by sample type.

The interpretation of the DmodX plot is exactly as for the strong sample outliers though, with any deviation from a consistent value for all SR or LTR samples worthy of further investigation.

### Loadings plots

The PCA loadings represent the weight of each original feature in forming the new PCA variables (scores). Thus there is a set of loadings (with values corresponding to each original feature) for each component in the model. The loadings can be plotted as for the scores, in pairs, however for improved interpretation here the loadings are plotted for each component separately. For LC-MS data, this takes the form of an ion map with points coloured by the model loadings (Figure 5A); for NMR data a standard (the median) spectrum, with each point coloured by its relative contribution to that particular component (Figure 5B); and for targeted datasets points corresponding to each named feature (Figure 5C).

Fig. 5: Figure 5A. PCA loadings (LC-MS datasets).

Fig. 6: Figure 5B. PCA loadings (NMR datasets).

These plots are useful in showing the regions of the spectrum with the most variance, and, by comparison with the scores plots (Figures 2 and 9-11) useful in determining any relationship to analytical sources (for example, variance in QC samples or association with specific analytical parameters).

### Distributions of analytical parameters

Distributions of the sample metadata (in this case relating to recorded analytical parameters) are plotted in Figure 6 (in this example for the DEVSET LC-MS dataset).

This allows the user to check if the behaviour of each parameter is as expected.

### Heatmap of potential associations between analytical parameters and the main sources of variance

Potential associations between the principal components and any analytical parameters is tested by calculating either the correlation (for continuous data) or a Kruskal-Wallis test (for categorical data) between each parameter and the PCA scores for each component. The returned values are displayed in Figures 7 and 8, for continuous and categorical data respectively.

These allow a quick assessment of whether any of the largest sources of variance in a dataset may have analytical sources. By default, any significant associations (correlation > 0.3 or Kruskal-Wallis p-value < 0.05) are plotted (see below), these default values can be amended by setting the “r\_threshold” or “kw\_threshold” when calling ‘multivariateReport’.

### Scores plots coloured by analytical parameters with potential association

Additionally for any fields where the correlation or p-value (respectively) exceed the threshold (default thresholds *r\_threshold=0.3*, *kw\_threshold=0.05*) the PCA scores plots are generated with sample points coloured according to their values for the flagged analytical parameter (in this case *Run Order*):

This allows quick identification and assessment of any analytical or pre-processing issues, and the subsequent action required depends on the analytical parameter flagged, for example, in this case batch and run-order correction would need to be applied, as there is a strong association with run order in PC3.

### Further options

As for the main reports (see [Quality Assessment Reports](#)), there are a number of different options which can be specified by the user if required, some examples of which include:

Fig. 7: Figure 5C. PCA loadings (Targeted datasets).

Fig. 8: Figure 6. Histograms of metadata distributions (plotted for fields with non-uniform values only).

```
# To save as html documents with static images to a specified location ('saveDir'):
saveDir = '/path to save outputs'
nPYc.reports.multivariateReport(dataset, PCAmodel, destinationPath=saveDir)

# To report on only those samples and features not set to be masked from the dataset,
↳NOTE, PCAmodel must be generated with the same exclusion set:
PCAmodel = nPYc.multivariate.exploratoryAnalysisPCA(dataset, scaling=1,
↳withExclusions=True)
nPYc.reports.multivariateReport(dataset, PCAmodel, withExclusions=True)

# To plot scores plots coloured by any analytical parameter with a correlation
↳exceeding 0.4 (default value=0.3, see Figure 9 above):
nPYc.reports.multivariateReport(dataset, PCAmodel, r_threshold=0.4)
```

Full function parameters (which may be of interest to advanced users) are as follows:

#### **class** nPYc.reports.multivariateReport

PCA based analysis of a dataset. A PCA model is generated for the data object, then potential associations between the scores and any sample metadata determined by correlation (continuous data) or a Kruskal-Wallis test (categorical data).

The multivariateReport has three options for the **reportType** argument:

- **‘analytical’** Reports on analytical qualities of the data only (as defined in the relevant SOP).
- **‘biological’** Reports on biological qualities of the data only (all columns in *sampleMetadata* except those defined as analytical or skipped in the SOP).
- **‘all’** Reports on all qualities of the data (all columns in *sampleMetadata* except those defined as skipped in the SOP).

#### **Parameters**

- **dataTrue** (*Dataset*) – Dataset to report on
- **pcaModel** (*ChemometricsPCA*) – PCA model object (scikit-learn based)
- **reportType** (*str*) – Type of sample metadata to report on, one of analytical, biological or all
- **withExclusions** (*bool*) – If True, only report on features and samples not masked by the sample and feature masks
- **biologicalMeasurements** (*dict*) – Dictionary of type of data contained in each biological sampleMetadata field. Keys are sampleMetadata column names, and values one of ‘categorical’, ‘continuous’, ‘date’
- **dModX\_criticalVal** (*None or float*) – Samples with a value in DModX space exceeding this critical value are listed as potential outliers

Fig. 9: Figure 7. Heatmap of correlation to PCA scores for suitable metadata fields.

Fig. 10: Figure 9. PCA scores plots coloured by metadata (significance by correlation).

- **dModX\_criticalVal\_type** (*None or str*) – Type of critical value in DModX, one of `Fcrit` or `Percentile`
- **scores\_criticalVal** (*None or float*) – Samples with a value in scores space exceeding this critical value are listed as potential outliers
- **kw\_threshold** (*None or float*) – Fields with a Kruskal-Willis p-value greater than this are not deemed to have a significant association with the PCA score
- **r\_threshold** (*None or float*) – Fields with a (absolute) correlation coefficient value less than this are not deemed to have a significant association with the PCA score
- **hotellings\_alpha** (*float*) – Alpha value for plotting the Hotelling’s ellipse in scores plots (default = 0.05)
- **excludeFields** (*None or list*) – If not `None`, list of sample metadata fields to be additionally excluded from analysis
- **destinationPath** (*None or str*) – If `None` plot interactively, otherwise save report to the path specified

## 10.3 Interactive Plots

Scores and loadings from a *PCAModel* can also be explored interactively with the `plotScoresInteractive()` and `plotLoadingsInteractive()` functions.

For example, a scores plot for components 1 and 2, with sample points coloured by *Run Order* can be generated using:

```
data = nPYc.plotting.plotScoresInteractive(dataset, PCAModel, 'Run Order',  
↪ components=[1, 2])  
iplot(data)
```

Similarly a loadings plot for component 2 can be generated using:

```
data = nPYc.plotting.plotLoadingsInteractive(dataset, PCAModel, component=2)  
iplot(data)
```

Again, see the *Plot Gallery* for examples.

Dilution effects on global sample intensity can be normalised by attaching one of the classes in the *normalisation* sub-module to the *Normalisation* attribute of a *Dataset*.

By default new *Dataset* objects have a *NullNormaliser* attached, which carries out no normalisation. By assigning an instance of a *Normaliser* class all calls to *intensityData* to return values transformed by the normaliser. For example, to return total area normalised values:

```
totalAreaNormaliser = nPYc.utilities.normalisation.TotalAreaNormaliser()  
dataset.Normalisation = totalAreaNormaliser
```

There are three built-in normalisation objects:

- Null normaliser (*NullNormaliser*): no normalisation performed
- Probabilistic quotient normaliser (*ProbabilisticQuotientNormaliser*): performs probabilistic quotient normalisation (Dieterle *et al.*<sup>1</sup>)
- Total area normaliser (*TotalAreaNormaliser*): performs normalisation where each row (sample) is divided by the total sum of its variables (columns)

## 11.1 Normalisation Syntax and Parameters

The main function parameters (which may be of interest to advanced users) are as follows:

The *utilities* module implements several *Normaliser* objects, that perform intensity normalisation on the provided numpy matrix.

All normaliser objects must implement the *Normaliser* abstract base class.

Normalisers may be configured as required upon initialisation, then a normalised view of a matrix obtained by passing the data to be normalised to the *normalise()* method.

---

<sup>1</sup> Frank Dieterle, Alfred Ross, Götz Schlötterbeck and Hans Senn. Probabilistic quotient normalization as robust method to account for dilution of complex biological mixtures. application in <sup>1</sup>H NMR metabonomics. *Analytical Chemistry*, 78(13):4281 – 90, 2006. URL: <https://pubs.acs.org/doi/10.1021/ac051632c>

Once `normalise()` has been called, the normalisation coefficients last used can be obtained from `normalisation_coefficients`.

**class** nPYc.utilities.normalisation.NullNormaliser

Null normalisation object which performs no normalisation, returning the provided matrix unchanged when `normalise()` is called.

**normalisation\_coefficients**

Returns normalisation coefficients. :return: 1

**normalise**(X)

Returns **X** unchanged.

**Parameters** **X** (*numpy.ndarray*, shape [n\_samples, n\_features]) – Data intensity matrix

**Returns** The original **X** matrix without any modification

**Return type** *numpy.ndarray*, shape [n\_samples, n\_features]

**class** nPYc.utilities.normalisation.ProbabilisticQuotientNormaliser (*reference=None, referenceDescription=None*)

Normalisation object which performs Probabilistic Quotient normalisation (Dieterle *et al* Analytical Chemistry, 78(13):4281 – 90, 2006)

**Parameters**

- **reference** (*str*, *int*, or *numpy.ndarray*) – Source of the reference profile. If *None*, use the median of **X**, if an *int* treat as the index of a spectrum in **X** to use as the reference, if an array with same width as **X**, treat as the reference profile.
- **referenceDescription** (*None*, or *str*) – A textual description of the reference provided
- **keepMagnitude** (*bool*) – If *True* scales **X** such that the mean area of **X** remains constant for the dataset as a whole.

**normalisation\_coefficients**

Returns the last set of normalisation coefficients calculated.

**Returns** Normalisation coefficients or *None* if they have not been generated yet

**Raises** **AttributeError** – Setting the normalisation coefficients directly is not allowed and raises an error

**reference**

Allows the reference profile used to calculate fold-changes to be queried or set.

**Returns** The reference profile used to calculate normalisation coefficients

**normalise**(X)

Apply Probabilistic Quotient normalisation to a dataset.

**Parameters**

- **X** (*numpy.ndarray*, shape [n\_samples, n\_features]) – Data intensity matrix
- **reference** (*numpy.ndarray*, shape [n\_features]) – Spectrum to use as the normalisation reference

**Returns** A read-only, normalised view of **X**

**Return type** `numpy.ndarray`, shape `[n_samples, n_features]`

**Raises** **ValueError** – if `X` is not a `numpy` 2-d array representing a data matrix

**class** `nPYc.utilities.normalisation.TotalAreaNormaliser` (*keepMagnitude=True*)

Normalisation object which performs Total Area normalisation. Each row in the matrix provided will be scaled to sum to the same value.

**Parameters** **keepMagnitude** (*bool*) – If `True` scales `X` such that the mean area of `X` remains constant for the dataset as a whole.

**normalisation\_coefficients**

Returns the last set of normalisation coefficients calculated.

**Returns** Normalisation coefficients or `None` if they have not been generated yet

**Raises** **AttributeError** – Setting the normalisation coefficients directly is not allowed and raises an error

**normalise** (`X`)

Apply Total Area normalisation to the dataset.

**Parameters** **X** (*numpy.ndarray*, shape `[n_samples, n_features]`) – Data intensity matrix

**Returns** A read-only, normalised view of `X`

**Return type** `numpy.ndarray`, shape `[n_samples, n_features]`

**Raises** **ValueError** – If `X` is not a `numpy` 2-d array representing a data matrix





## CHAPTER 12

---

### Exporting Data

---

Datasets can be exported in a variety of formats with the `exportDataset()` method.

The default export (`saveFormat=CSV`) results in production of three separate CSV files:

```
saveDir = '/path to save outputs'
dataset.exportDataset(destinationPath=saveDir)
```

- `sampleMetadata`: with a row for every sample and a column for every separate sample-related metadata field
- `featureMetadata`: with a row for every feature and a column for each separate feature-related metadata field
- `intensityData`: intensity value per variable (column) and sample (row)

An alternative option (`saveFormat=UnifiedCSV`) results in export of a single file, which contains the `sampleMetadata`, `featureMetadata`, and `intensityData` concatenated together, with samples in rows, and features in columns:

```
dataset.exportDataset(saveFormat='UnifiedCSV', destinationPath=saveDir)
```

The nPYc-Toolbox also supports exporting metadata in ISATAB format.

Reports can also be saved to file, see [Quality Assessment Reports](#) for details.



## 13.1 Built-in Configuration SOPs

The following tables list, define and give default values for all of the SOP parameters for each method.

### 13.1.1 All Dataset Objects

Table 1: Required SOP parameters for all Dataset objects

Key	Type	Default value	Role
'noFiles'	int	10	When showing a ranked list of files show only the top/bottom <i>noFiles</i>
'dpi'	int	300	Raster resolution when plotting figures
'figureSize'	list of float	[11,7]	Size to plot figures
'figureFormat'	str	'png'	Format to save figures in
'histBins'	int	100	Number of bins to use when drawing histograms
'quantiles'	list of float	[25, 75]	When calculating percentiles, use this default range

Table 2: Optional SOP parameters for all Dataset objects

Key	Type	De- fault value	Role
‘analyticalMeasurements’	dict	{ }	Set of key, value pairs where each key is a column in <i>sampleMetadata</i> and the value is ‘categorical’ or ‘continuous’ depending on parameter type. Columns described here will be checked for association in the multivariate quality control reports when run with the analytical setting.
‘excludeFromPlotting’	list of str	[ ]	Column names in <i>sampleMetadata</i> to exclude from plotting
‘sampleMetadataNotExported’	list of str	[“Ex- clu- sion De- tails”]	Column names in <i>sampleMetadata</i> to exclude from data export
‘featureMetadataNotExported’	list of str	[ ]	Column names in <i>featureMetadata</i> to exclude from data export

### 13.1.2 MSDataset Objects

Table 3: SOP parameters for all MSDataset objects

Parameter	Type	Default value	Role
‘corrThreshold’	float	0.7	When filtering features by correlation to dilution using the <i>Serial Dilution Sample</i> , the correlation must be above this
‘corrMethod’	str	‘pearson’	Type of correlation to linearity to calculate, must be ‘pearson’ or ‘spearman’
‘rsdThreshold’	float	30	When filtering features by <i>RSD</i> , the RSD must be below this
‘varianceRatio’	float	1.1	When filtering features RSD in Study Samples must be at least RSD in Precision Reference * this value
‘blankThreshold’	float	1.1	When filtering features RSD in Study Samples must be at least RSD in Procedural Blank samples * this value
‘artifactualFilter’	str (bool)	‘False’	Flag for whether artifactual filtering should be applied when filtering features
‘deltaMzArtifactual’	float	0.1	‘artifactualFilter’ parameter: Maximum allowed m/z distance between two grouped features
‘overlapThresholdArtifactual’	int	50	‘artifactualFilter’ parameter: Minimum peak overlap between two grouped features
‘corrThresholdArtifactual’	float	0.9	‘artifactualFilter’ parameter: Minimum correlation between two grouped features
‘filenameSpec’	str (regex)	see ‘Gener- icMS.json’	Regular expression to pull out information from raw MS data filenames (as per NPC standard naming conventions)

### 13.1.3 NMRDataset Objects

Table 4: SOP parameters for all NMRDataset objects

Parameter	Type	Default value (for GenericNMR-Urine)	Default value (for GenericNMR-Blood)	Role
'bounds'	list of float	[-1, 10]	[-1, 10]	Region of the original spectrum to re-interpolate and retain
'variable-Size'	int	20000	20000	Number of points in the re-interpolated spectrum
'alignTo'	str	'singlet'	'doublet'	Type of signal to calibrate to
'calibrateTo'	float	0	5.233	Chemical shift value to calibrate to
'ppm-SearchRange'	list of float	[-0.3, 0.3]	[4.9, 5.733]	Chemical shift region to search for calibration signal
'LWpeak-Multiplicity'	str	'singlet'	'quartet'	Type of signal used to measure line width
'LW-peakRange'	list of float	[-0.1, 0.1]	[4.08, 4.14]	Chemical shift region to search for line width signal
'LW-FailThreshold'	float	1.3	1.3	Line-width check cut-off in Hz
'base-lineCheck-Region'	list of list pairs of floats	[[[-2, -0.5], [9.5, 12.5]]]	[[[-2, -0.5], [9.5, 12.5]]]	Chemical shift regions to use in base-line quality checks
'solvent-PeakCheck-Region'	list of list pairs of floats	[[[4.6, 4.7],[4.9,5]]]	[[[4.4, 4.5], [4.85,5]]]	Chemical shift regions to use in water suppression quality checks
'exclusion-Regions'	list of list pairs of floats	[[[-0.2,0.2],[4.7,4.9]]]	[[[-0.2,0.2],[4.5,4.85]]]	Chemical shift regions to mark for exclusion by default during pre-processing

## 13.2 Generation of Targeted SOPs

To create a new pre-defined TargetLynx SOP (`fileType == 'TargetLynx'`) the *JSON* SOP must contain the following fields, the list must cover all compounds in the same order:

- **methodName** The name of the method.
- **chromatography** The chromatography employed.
- **ionisation** The polarity employed.
- **compoundID** A list of numeric ID ("1","2",...) that matches the TargetLynx compound ID.
- **compoundName** A list of compound names.
- **IS** A list of "True" "False" to indicate if the compound is an Internal Standard.
- **unitFinal** A list of the compound measurement unit after application of the *unitCorrectionFactor* (can be left blank "").
- **unitCorrectionFactor** A list of values by which to multiply the measured concentration in order to reach the *unitFinal* ("1","0.1","1000").

- **calibrationMethod** A list of the calibration method employed for the compound, from enum CalibrationMethod: "noIS" for compounds without Internal Standard (and Internal Standards themselves) = (use area), "backcalculatedIS" for compounds using an Internal Standard = (use response), "noCalibration" for compounds not quantified (Monitored for relative information).

- **calibrationEquation**

A list of equations to obtain the concentration given *area*, *responseFactor*, *a* and *b*. *responseFactor* = (IS conc/IS Area)=*response*/Area (for noIS, *responseFactor* will be 1) is automatically estimated from calibration samples.

The calibration equation is only employed if values <LLOQ are replaced by the noise level (*Targeted-Dataset.\_targetLynxApplyLimitsOfQuantificationNoiseFilled()*)

**Calibration curve in TargetLynx is defined/established as:  $\text{response} = a * \text{concentration} + b$  (eq. 1)**

- response is defined as:  $\text{response} = \text{Area} * (\text{IS conc} / \text{IS Area})$  (eq. 2) [for 'noIS' response = Area]
- using eq. 2, we can approximate the ratio IS Conc/IS Area in a representative sample as:  $\text{responseFactor} = \text{response} / \text{area}$  (eq. 3)
- Therefore:  $\text{concentration} = ((\text{area} * \text{responseFactor}) - b) / a$  (eq. 4)
- If in TargetLynx 'axis transformation' is set to log (but still use 'Polynomial Type'=linear and 'Fit Weighting'=None)
- eq.1 is changed to:  $\log(\text{response}) = a * \log(\text{concentration}) + b$  (eq. 5)
- and eq. 4 changed to:  $\text{concentration} = 10^{((\log(\text{area} * \text{responseFactor}) - b) / a)}$  (eq. 5)
- Examples: `"((area * responseFactor)-b)/a"`, `"10**((numpy.log10(area * responseFactor)-b)/a)"`, `"area/a"` | if b not needed, set to 0 in csv [use for linear noIS, area=response, responseFactor=1, and response = a \* concentration]

- **quantificationType** A list of the type of quantification employed, from enum QuantificationType: "IS", "QuantOwnLabeledAnalogue", "QuantAltLabeledAnalogue", "QuantOther" or "Monitored"

---

**Note:** *quantificationType* "IS" must match with *IS* "True". *quantificationType* "Monitored" must match with *calibrationMethod* "noCalibration".

---

- **externalID** A list of external ID, each external ID must also be present as its own field as a list of identifier (for that external ID). For example, if `"externalID": ["PubChem ID"]`, the field `"PubChem ID": ["ID1", "ID2", "", "ID75"]` is required.
- **sampleMetadataNotExported** A list of sampleMetadata columns to exclude from export and reports.
- **featureMetadataNotExported** A list of featureMetadata columns to exclude from export and reports.

Listing 1: Example TargetLynx SOP for the Amino Acids assay (Gray N. *et al.* Human Plasma and Serum via Precolumn Derivatization with 6-Aminoquinolyl-N-hydroxysuccinimidyl Carbamate: Application to Acetaminophen-Induced Liver Failure. *Analytical Chemistry*, 89, 2017, 24782487)

```
{
  "methodName": "NPC LC-MS Targeted - Amino Acid",
  "chromatography": "R",
  "ionisation": "POS",
  "compoundID": ["1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13", "14",
  ↪ "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30", "31",
  ↪ "32", "33", "34", "35", "36", "37", "38", "39", "40", "41", "42", "43", "44", "45", "46", "47",
  ↪ "48", "49", "50", "51", "52", "53", "54", "55", "56", "57", "58", "59", "60", "61", "62", "63", "64",
  ↪ "65", "66", "67", "68", "69", "70", "71", "72", "73"],
  "compoundName": ["4-Hydroxyproline", "Alanine", "Alanine-13C3-15N", "Arginine",
  ↪ "Arginine-13C6-15N4", "Aspartic acid", "Aspartic acid-13C4-15N", "Asparagine",
  ↪ "Carnosine", "Cystine", "Ethanolamine", "Glutamic acid", "Glutamic acid-13C5-15N",
  ↪ "Glutamine", "Glutamine-13C5", "Glycine", "Glycine-13C2-15N", "Histidine", "Histidine-
  ↪ 13C6-15N3", "Isoleucine", "Isoleucine-13C6-15N", "Leucine", "Leucine-13C6-15N", "Lysine",
  ↪ "Lysine-13C6-15N2", "Methionine", "Methionine-13C5-15N", "Phenylalanine",
  ↪ "Phenylalanine-13C9-15N", "Proline", "Proline-13C5-15N", "Serine", "Serine-13C3-15N",
  ↪ "Threonine", "Threonine-13C4-15N", "Tryptophan", "Tyrosine", "Valine", "Valine-13C5-15N",
  ↪ "beta-Amino-iso-Butyric acid", "Citrulline", "Cystathionine", "3-Methylhistidine", "1-
  ↪ Methylhistidine", "Homoserine", "Hydroxylysine", "Ornithine", "Aminoadipic acid", "alpha-
  ↪ Amino-n-Butyric acid", "Phosphoserine", "Sarcosine", "Taurine", "beta-Alanine", "gamma-
  ↪ Amino-n-Butyric acid", "Methylamine", "Creatine", "4-Aminohippuric acid", "5-
  ↪ Aminovaleric acid", "Allantoin", "Anserine", "Cysteine", "Epinephrine", "Galactosamine",
  ↪ "Glutathione", "Homocystine", "N-methyl-L-phenylalanine", "N-methyl-valine", "Octopamine",
  ↪ "Putrescine", "Tryptamine", "Tyramine", "Tyrosine-13C9-15N", "Glycylglycine"],
  "IS": ["False", "False", "True", "False", "True", "False", "True", "False", "True", "False", "False",
  ↪ "False", "False", "False", "True", "False", "True", "False", "True", "False", "True", "False",
  ↪ "True", "False", "True", "False", "True", "False", "True", "False", "True", "False", "True",
  ↪ "False", "True", "False", "True", "False", "False", "False", "True", "False", "False", "False",
  ↪ "False", "False", "False", "False", "False", "False", "False", "False", "False", "False", "False",
  ↪ "False", "False", "False", "False", "False", "False", "False", "False", "False", "False",
  ↪ "False", "False", "False", "False", "False", "False", "False", "False", "False", "True",
  ↪ "False"],
  "unitFinal": ["\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M",
  ↪ "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M",
  ↪ "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M",
  ↪ "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M",
  ↪ "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M",
  ↪ "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M",
  ↪ "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M", "\u00B5M",
  ↪ "noUnit", "noUnit", "noUnit", "noUnit", "noUnit", "noUnit", "noUnit", "noUnit", "noUnit",
  ↪ "noUnit", "noUnit", "noUnit", "noUnit", "noUnit", "noUnit", "noUnit", "noUnit", "\u00B5M",
  ↪ "noUnit"],
  "unitCorrectionFactor": ["1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1",
  ↪ "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1",
  ↪ "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1",
  ↪ "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1", "1"],
  "calibrationMethod": ["backcalculatedIS", "backcalculatedIS", "noIS",
  ↪ "backcalculatedIS", "noIS", "backcalculatedIS", "noIS", "backcalculatedIS",
  ↪ "backcalculatedIS", "backcalculatedIS", "backcalculatedIS", "backcalculatedIS", "noIS",
  ↪ "backcalculatedIS", "noIS", "backcalculatedIS", "noIS", "backcalculatedIS", "noIS",
  ↪ "backcalculatedIS", "noIS", "backcalculatedIS", "noIS", "backcalculatedIS", "noIS",
  ↪ "backcalculatedIS", "noIS", "backcalculatedIS", "noIS", "backcalculatedIS", "noIS",
  ↪ "backcalculatedIS", "noIS", "backcalculatedIS", "noIS", "backcalculatedIS", "noIS",
  ↪ "backcalculatedIS", "noIS", "backcalculatedIS", "noIS", "backcalculatedIS",
  ↪ "backcalculatedIS", "backcalculatedIS", "noIS", "backcalculatedIS", "backcalculatedIS",
  ↪ "backcalculatedIS", "backcalculatedIS", "backcalculatedIS", "backcalculatedIS",
  ↪ "backcalculatedIS", "backcalculatedIS", "backcalculatedIS", "backcalculatedIS",
  ↪ "backcalculatedIS", "noCalibration", "noCalibration", "noCalibration", "noCalibration",
  ↪ "noCalibration", "noCalibration", "noCalibration", "noCalibration", "noCalibration"]
}
```

(continued from previous page)

```

"calibrationEquation": ["((area * responseFactor)-b)/a", "((area *
→responseFactor)-b)/a", "", "((area * responseFactor)-b)/a", "", "((area *
→responseFactor)-b)/a", "", "((area * responseFactor)-b)/a", "((area * responseFactor)-
→b)/a", "((area * responseFactor)-b)/a", "((area * responseFactor)-b)/a", "((area *
→responseFactor)-b)/a", "", "((area * responseFactor)-b)/a", "", "((area *
→responseFactor)-b)/a", "", "((area * responseFactor)-b)/a", "", "((area *
→responseFactor)-b)/a", "", "((area * responseFactor)-b)/a", "", "((area *
→responseFactor)-b)/a", "", "((area * responseFactor)-b)/a", "", "((area *
→responseFactor)-b)/a", "", "((area * responseFactor)-b)/a", "", "((area *
→responseFactor)-b)/a", "", "((area * responseFactor)-b)/a", "", "((area *
→responseFactor)-b)/a", "((area * responseFactor)-b)/a", "((area * responseFactor)-b)/a",
→", "", "((area * responseFactor)-b)/a", "((area * responseFactor)-b)/a", "((area *
→responseFactor)-b)/a", "((area * responseFactor)-b)/a", "((area * responseFactor)-b)/a",
→", "((area * responseFactor)-b)/a", "((area * responseFactor)-b)/a", "((area *
→responseFactor)-b)/a", "((area * responseFactor)-b)/a", "((area * responseFactor)-b)/a",
→", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "", "",
→", "quantificationType": ["QuantAltLabeledAnalogue", "QuantOwnLabeledAnalogue", "IS
→", "QuantOwnLabeledAnalogue", "IS", "QuantOwnLabeledAnalogue", "IS",
→"QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue",
→"QuantAltLabeledAnalogue", "QuantOwnLabeledAnalogue", "IS", "QuantOwnLabeledAnalogue",
→"IS", "QuantOwnLabeledAnalogue", "IS", "QuantOwnLabeledAnalogue", "IS",
→"QuantOwnLabeledAnalogue", "IS", "QuantOwnLabeledAnalogue", "IS",
→"QuantOwnLabeledAnalogue", "IS", "QuantOwnLabeledAnalogue", "IS",
→"QuantOwnLabeledAnalogue", "IS", "QuantOwnLabeledAnalogue", "IS",
→"QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue", "QuantOwnLabeledAnalogue", "IS",
→"QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue",
→"QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue",
→"QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue",
→"QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue", "QuantAltLabeledAnalogue",
→"Monitored", "Monitored", "Monitored", "Monitored", "Monitored", "Monitored", "Monitored",
→"Monitored", "Monitored", "Monitored", "Monitored", "Monitored", "Monitored", "Monitored",
→"Monitored", "Monitored", "Monitored", "IS", "Monitored"],
"externalID": [],
"sampleMetadataNotExported": ["TargetLynx Sample ID", "MassLynx Row ID", "Acqu
→Date", "Acqu Time", "Instrument", "Sample Type"],
"featureMetadataNotExported": ["calibrationEquation", "TargetLynx Feature ID",
→"unitCorrectionFactor", "CpdInfo", "Cpd Info", "Noise (area)", "a", "b", "r", "r2",
→"r^2", "# left", "Excluded"],
"analyticalMeasurements": {"Study" : "categorical", "Instrument" : "categorical
→", "Re-Run" : "categorical", "Supplemental Injections" : "categorical", "Matrix" :
→"categorical", "Well" : "categorical", "Plate" : "categorical", "Batch" :
→"categorical", "Dilution" : "continuous", "Measurement Date" : "date", "Measurement
→Time" : "date", "Acquired Time" : "date", "Run Order" : "continuous", "Correction
→Batch" : "categorical", "Assay data name": "categorical", "Assay data location":
→"categorical", "Sample position": "categorical", "Sample batch": "categorical",
→"Acquisition batch": "categorical", "Plot Sample Type": "categorical", "AssayRole":
→"categorical", "SampleType": "categorical", "Exclusion Details": "categorical",
→"Skipped": "categorical", "Assay protocol": "categorical"},
"excludeFromPlotting": ["Sample File Name", "Sample Base Name", "Batch Termini
→", "Study Reference", "Long-Term Reference", "Method Reference", "Dilution Series",
→"Skipped", "Study Sample", "File Path", "Exclusion Details", "Assay protocol",
→"Status", "Measurement Date", "Measurement Time", "Data Present", "LIMS Present",
→"LIMS Marked Missing", "Assay data name", "Assay data location", "AssayRole",
→"SampleType", "Sampling ID", "Plot Sample Type", "SubjectInfoData", "Detector Unit",
→"TargetLynx Sample ID", "MassLynx Row ID"]

```



(continued from previous page)

}

Behaviour of many aspects of the toolbox can be modified in a repeatable manner to create new workflows by creating configuration files.

Default parameters for *Dataset* objects can be configured as they are initialised by supplying a SOP file containing the desired parameters.

SOP parameters include aesthetic factors such as figure sizes and formats as well as quality control and analytical parameters.

By default SOPs are read from the `nPYc/StudyDesigns/SOP/` directory, but this can be overridden by the directory specified in `sopPath=`, that will be searched before the built in SOP directory.

SOP files are simple **JSON** format files, whose contents are used to populate the *Attributes* dictionary. See the default files in `nPYc/StudyDesigns/SOP/` for examples.

The nPYc-Toolbox comes with a built-in set of configuration SOP files for each dataset type, for full details see *Built-in Configuration SOPs*.

New pre-defined TargetLynx SOP files can also be created, for full details see *Generation of Targeted SOPs*.



This section describes the main enumerations function parameters (which may be of interest to advanced users).

The `enumerations` module provides a set of enumerations (complete listings of all possible items in a collection) for common types referenced in profiling experiments.

**class** `nPYc.enumerations.VariableType`

Enumeration of data sampling modalities.

- *Discreetly* sampled data types are those where the adjacency of variables is unimportant
- In *spectral, continuum, and synonymous data*, the ordering of variables is important in the interpretation of the data

**Discrete** = 1

**Continuum** = 2

**Spectral** = 2

**Profile** = 2

**class** `nPYc.enumerations.Ionisation`

Enumeration of ionisation methods in Mass Spectrometry.

**EI** = 'Electron Impact'

**ESI** = 'Electrospray Ionisation'

**DESI** = 'Desorption Electrospray Ionisation'

**MALDI** = 'Matrix Assisted Laser Desorption Ionisation'

**class** `nPYc.enumerations.SampleType`

Enumeration of distinct sample types.

- *Study Sample* comprise the study in question
- *Study Pool* consists of a mixture of all *study samples*

- *External Reference* a sample of a comparable *matrix* to the *Study Samples*, but not a sample (or mixture) derived from samples acquired as part of the study. Acquired, for example, for assessing analytical quality between studies.
- *Method Reference* consists of a synthetic mixture of known chemical standards
- *Procedural Blank* a blank sample not expected to contain any signals from the sample matrix

**StudySample** = 'Study Sample'

**StudyPool** = 'Study Pool'

**ExternalReference** = 'External Reference'

**MethodReference** = 'Method Reference'

**ProceduralBlank** = 'Procedural Blank'

**class** nPYc.enumerations.DatasetLevel

An enumeration.

**Unknown** = 0

**Empty** = 1

**ValuesOnly** = 2

**QCReady** = 3

**class** nPYc.enumerations.AssayRole

Enumeration of assay roles.

- *Assay* form the core of an analysis
- *Precision Reference* repeatedly acquired, and used to calculate measures of analytical precision
- *Linearity Reference* used to assess the linearity of response in the dataset, often by repeated injection at varying concentrations or levels of dilution. If generated by dilution from a base-level, the percentage concentration of each Linearity Reference sample is indicated in the 'Dilution' field of the *sampleMetadata* table

**Assay** = 'Assay'

**PrecisionReference** = 'Precision Reference'

**LinearityReference** = 'Linearity Reference'

**Blank** = 'Blank'

**class** nPYc.enumerations.Polarity

Enumeration of ionisation polarity.

**Positive** = 1

**Negative** = 2

**class** nPYc.enumerations.QuantificationType

Enumeration of quantification types.

- *IS* for Internal Standards
- *QuantOwnLabeledAnalogue* for compounds quantified and validated with own labeled analogue
- *QuantAltLabeledAnalogue* for compounds quantified and validated with alternative labeled analogue
- *QuantOther* for compounds quantified using another method
- *Monitored* for compounds monitored for relative information

- *BrukerivDrQuant* for compounds quantified with Bruker ivDr methods
- *BrukerivDrEstimate* for compounds estimated with Bruker ivDr methods

**IS** = 'Internal Standard'

**QuantOwnLabeledAnalogue** = 'Quantified and validated with own labeled analogue'

**QuantAltLabeledAnalogue** = 'Quantified and validated with alternative labeled analogue'

**QuantOther** = 'Other quantification'

**Monitored** = 'Monitored for relative information'

**BrukerivDrQuant** = 'Quantified using Bruker Biospin ivDr methods'

**BrukerivDrEstimate** = 'Estimated from other parameters using Bruker Biospin ivDr method'

**class** nPYc.enumerations.CalibrationMethod

Enumeration of distinct calibration methods.

- *noCalibration* for compounds not quantified (monitored for relative information)
- *noIS* for compounds without Internal Standard (and Internal Standards themselves)
- *backcalculatedIS* for compounds using an Internal Standard
- *otherCalibration* for compounds employing another calibration approach
- *nmrCalibration* for compounds quantified by NMR

**noCalibration** = 'No calibration'

**noIS** = 'No Internal Standard'

**backcalculatedIS** = 'Backcalculated with Internal Standard'

**otherCalibration** = 'Other calibration method'

**nmrCalibration** = 'NMR quantitation'

**class** nPYc.enumerations.AnalyticalPlatform

Enumeration of analytical platform types.

- *NMR* for Nuclear Magnetic Resonance Spectroscopy
- *MS* for Mass Spectrometry
- *Other* placeholder for any other type of instrumentation

**NMR** = 'NuclearMagneticResonanceSpectroscopy'

**MS** = 'MassSpectrometry'

**Other** = 'Other'



## Utility Functions

This section describes the main utility function parameters (which may be of interest to advanced users).

The *utilities* module provides convenience functions for working with profiling datasets.

`nPYc.utilities.rsdc(data)`

Calculate percentage *relative standard deviation* for each column in *data*.

$$rsd(x) = \frac{\sigma_x}{\mu_x} \times 100$$

Where RSDs cannot be calculated, (i.e. means of zero), `numpy.finfo(numpy.float64).max` is returned.

**Parameters** *data* (*numpy.ndarray*) – *n* by *m* numpy array of data, with features in columns, and samples in rows

**Returns** *m* vector of RSDs

**Return type** *numpy.ndarray*

`nPYc.utilities.buildFileList(filepath, pattern)`

Search for data files, by attempting to match to the file path regex *pattern*. :param filepath: Look for data in all the directories under this location :type searchDirectory: str :param pattern: Recognise experimental data by matching path to this compiled regex :type pattern: re.SRE\_Pattern :return: A list of all paths below *searchDirectory* that matched *pattern* :rtype: list[str,]

`nPYc.utilities.sequentialPrecision(data)`

Calculate percentage sequential precision for each column in *data*. Sequential precision for feature *x* is defined as:

$$sp(x) = \frac{\sqrt{\frac{1}{n-1} \sum_{i=1}^{n-1} (x_{i+1} - x_i)^2} / 2}{\mu_x} \times 100$$

**Parameters** *data* (*numpy.ndarray*) – *n* by *m* numpy array of measures, with features in columns, and samples in rows

**Returns** *m* vector of sequential precision measures

**Return type** *numpy.ndarray*

`nPYc.utilities.rsdsBySampleType` (*dataset*, *onlyPrecisionReferences=True*, *useColumn='SampleType'*)

Return percent RSDs calculated for the distinct class values in *useColumn*, defaults to the *SampleType* enums in 'SampleType'.

**Parameters**

- **dataset** (*Dataset*) – Dataset object to generate RSDs for.
- **onlyPrecisionReferences** (*bool*) – If *True* only use samples with the 'Assay-Role' *PrecisionReference*

**Returns** Dict of RSDs for each group

**Return type** dict(str:numpy array)



---

## Plotting Functions

---

Throughout the toolbox, care has been taken to present results and project analyses into outputs of a form directly interpretable by the analyst, such as projecting the loadings of a multivariate model onto the spectrum in the case of NMR, and as an ion-map for LC-MS.

See the *Plot Gallery* for a visual overview of the available plots.

The *plotting* module contains function to generate several common visualisations.

Plots are built upon *seaborn* for aesthetics, or when interactivity is required, *plotly*.

Most plots support a set of common configuration parameters to allow customisation of various display options. Common parameters that may be specified as keyword arguments are:

**plottingFunctions(\*vars, \*\*kwargs):**

### Parameters

- **savePath** (*str*) – If *None* plot interactively, otherwise save the figure to the path specified
- **figureFormat** (*str*) – If saving the plot, use this format
- **dpi** (*int*) – Plot resolution
- **figureSize** (*tuple(float, float)*) – Dimensions of the figure

Interactive plots utilise the *plotly* framework to provide controls, when using *plotly* you should ensure that the environment is configured according to the instructions at [Offline Plots in Plotly in Python](#)

```
nPYc.plotting.histogram(values, inclusionVector=None, quantiles=None, histBins=100,  
                        color=None, logy=False, logx=False, **kwargs)
```

Plot a histogram of values, optionally segmented according to observed quantiles.

Quantiles can be calculated on a second *inclusionVector* when specified.

### Parameters

- **values** (*numpy.array or dict*) – Values to plot

- **inclusionVector** (*None or numpy.array*) – Optional second vector with same size as values, used to select quantiles for plotting.
- **quantiles** (*None or List*) – List of quantile bounds to segment the histogram by
- **title** (*str*) – Title for the plot
- **xlabel** (*str*) – Label for the X-axis
- **histBins** (*int*) – Number of bins to break the histogram into
- **color** (*None or List*) – List of specific colours to use for plotting
- **logy** (*bool*) – If True plot y on a log scale
- **logx** (*bool*) – If True plot x on a log scale
- **xlim** (*tuple of int*) – Specify upper and lower bounds of the X axis

```
nPYc.plotting.plotBatchAndROCorrection(msData, msDatacorrected, featureList, addViolin=True, sampleAnnotation=None, logy=False, title="", savePath=None, figureFormat='png', dpi=72, figureSize=(11, 7))
```

Visualise the run-order correction applied to features, by plotting the values before and after correction, along with the fit calculated.

#### Parameters

- **msData** (*MSDataset*) – Dataset prior to correction
- **msDatacorrected** (*MSDataset*) – Dataset post-correction
- **featureList** (*list[int, ]*) – List of ints specifying indices of features to plot
- **addViolin** (*bool*) – If true, plot distributions as violin plots in addition to the longitudinal trend
- **sampleAnnotation** (*dict*) – Samples for annotation in plot, must include fields 'rank': index (int) and 'id': sample name (str, as in `msData.sampleMetadata['Sample File Name']`). For example, item['AbundanceSamples'] in `featureID.py`.
- **logy** (*bool*) – If True plot intensities on a log10 scale
- **title** (*str*) – Text to title each plot with
- **savePath** (*None or str*) – If None plot interactively, otherwise save the figures to the path specified

```
nPYc.plotting.plotTIC(msData, addViolin=True, addBatchShading=False, addLineAtGaps=False, colourByDetectorVoltage=False, logy=False, title="", withExclusions=True, savePath=None, figureFormat='png', dpi=72, figureSize=(11, 7))
```

Visualise TIC for all or a subset of features coloured by either dilution value or detector voltage. With the option to shade by batch.

---

**Note:** `addViolin` and `colourByDetectorVoltage` are mutually exclusive.

---

#### Parameters

- **msData** (*MSDataset*) – Dataset object
- **addViolin** (*bool*) – If True adds violin plots of TIC distribution pre and post correction split by sample type
- **addBatchShading** (*bool*) – If True shades plot according to sample batch

- **addLineAtGaps** (*bool*) – If *True* adds line where acquisition time is greater than double the norm
- **colourByDetectorVoltage** (*bool*) – If *True* colours points by detector voltage, else colours by dilution
- **logy** (*bool*) – If *True* plot *y* on a log scale
- **title** (*str*) – Title for the plot
- **withExclusions** (*bool*) – If *False*, discard masked features from the sum
- **savePath** (*None* or *str*) – If *None* plot interactively, otherwise save the figure to the path specified
- **figureFormat** (*str*) – If saving the plot, use this format
- **dpi** (*int*) – Plot resolution
- **figureSize** (*tuple(float, float)*) – Dimensions of the figure

`nPYc.plotting.plotTICinteractive` (*msData*, *plottype*='Sample Type', *labelby*='Run Order', *withExclusions*=*True*)

Interactively visualise TIC (coloured by batch and sample type) with plotly, provides tooltips to allow identification of samples.

Plots may be of two types: \* **'Sample Type'** to plot by sample type and coloured by batch \* **'Linearity Reference'** to plot LR samples coloured by dilution

#### Parameters

- **msData** (*MSDataset*) – Dataset object
- **plottype** (*str*) – Select plot type, may be either *Sample Type* or *Linearity Reference*

**Returns** Data object to use with plotly

`nPYc.plotting.plotLRTIC` (*msData*, *sampleMask*=*None*, *colourByDetectorVoltage*=*False*, *title*="", *label*=*False*, *savePath*=*None*, *figureFormat*='png', *dpi*=72, *figureSize*=(11, 7))

Visualise TIC for linearity reference (LR) samples (either all or a subset) coloured by either dilution value or detector voltage.

#### Parameters

- **msData** (*MSDataset*) – Dataset object
- **sampleMask** (*None* or *array of bool*) – Defines subset of samples to plot, if *None* use *msData*'s built-in *sampleMask*
- **colourByDetectorVoltage** (*bool*) – If *True* colours points by detector voltage, else colours by dilution
- **title** (*str*) – Title for the plot
- **label** (*bool*) – If *True*, labels points with run order values
- **savePath** (*None* or *str*) – If *None*, plot interactively, otherwise attempt to save at this path.
- **format** (*str*) – Format to save figure
- **dpi** (*int*) – Resolution to draw at
- **figureSize** (*tuple(float, float)*) – Specify size of figure

```
nPYc.plotting.jointplotRSDvCorrelation(rsd, correlation, histBins=100, savePath=None, figureFormat='png', dpi=72, figureSize=(11, 7))
```

Plot a 2D histogram of feature RSDs vs correlations to dilution, with marginal histograms.

#### Parameters

- **rsd** (*numpy.array*) – Vector of feature relative standard deviations
- **correlation** (*numpy.array*) – Vector of correlation to dilution
- **histBins** (*int*) – Number of bins to break the histogram into
- **savePath** (*None* or *str*) – If *None*, plot interactively, otherwise attempt to save at this path
- **figureFormat** (*str*) – If saving the plot, use this format
- **dpi** (*int*) – Plot resolution
- **figureSize** (*tuple(float, float)*) – Dimensions of the figure

```
nPYc.plotting.plotCorrelationToLRbyFeature(msData, featureMask=None, title="", maxNo=5, savePath=None, figureFormat='png', dpi=72, figureSize=(11, 7))
```

Summary plots of correlation to dilution for a subset of features, separated by sample batch. Each figure includes a scatter plot of feature intensity vs dilution, TIC of LR and surrounding SP samples, and a heatmap of correlation to dilution for each LR batch subset, overall, and mean.

#### Parameters

- **msData** (*MSDataset*) – Dataset object
- **featureMask** (*None* or *array of bool*) – Limits plotting to a subset of features, if *None* use *msData*'s built-in *sampleMask*
- **title** (*str*) – Title for the plot
- **maxNo** (*int*) – Optional number of features to plot (default=10, i.e., 10 randomly selected features in *featureList* will be plotted)
- **savePath** (*None* or *str*) – If *None*, plot interactively, otherwise attempt to save at this path.
- **figureFormat** (*str*) – Format to save figure
- **dpi** (*int*) – Resolution to draw at
- **figureSize** (*tuple(float, float)*) – Specify size of figure

```
nPYc.plotting.plotIonMap(msData, useRetention=True, title=None, savePath=None, xlim=None, ylim=None, logx=False, logy=False, figureFormat='png', dpi=72, figureSize=(11, 7))
```

`plotIonMap(msData, **kwargs):`

Visualise features in a *MSDataset*, to visualise the features in terms of the raw data.

Plotting requires the presence of 'm/z' and 'Retention Time' columns in the *featureMetadata* table. If both 'm/z' and retention time are present, a 2D ion map is generated, otherwise a 1D mass-spectrum is plotted.

#### Parameters

- **msData** (*MSDataset*) – Dataset object to visualise
- **useRetention** (*bool*) – If *False* ignore any Retention Time information and plot a 1D mass spectrum

`nPYc.plotting.plotRSDs(dataset, ratio=False, savePath=None, color=None **kwargs)`

Visualise analytical *versus* biological variance.

Plot RSDs calculated in study-reference samples (analytical variance), versus those calculated in study samples (biological variance). RSDs can be visualised either in absolute terms, or as a ratio to analytical variation (*ratio=True*).

`plotRSDs()` requires that the dataset have at least two samples with the *PrecisionReference assay role*, if present, RSDs calculated on independent sets of *PrecisionReference* samples will also be plotted.

#### Parameters

- **dataset** (*Dataset*) – Dataset object to plot, the object must have greater than one ‘Study Sample’ and ‘Study-Reference Sample’ defined
- **ratio** (*bool*) – If *True* plot the ratio of analytical variance to biological variance instead of raw values
- **featureName** (*str*) – featureMetadata column name by which to label features
- **logx** (*bool*) – If *True* plot RSDs on a log10 scaled axis
- **xlim** (*None or tuple(float, float)*) – Tuple of (min, max) RSD values to plot
- **hLines** (*None or list*) – None or list of y positions at which to plot an horizontal line. Features are positioned from 1 to nFeat
- **savePath** (*None or str*) – If *None* plot interactively, otherwise save the figure to the path specified
- **color** (*None or seaborn.palettes.\_ColorPalette*) – Allows the default colour pallet to be overridden
- **featName** (*bool*) – If *True* y-axis label is the feature Name, if *False* features are numbered.

`nPYc.plotting.plotRSDsInteractive(dataset, featureName='Feature Name', ratio=False, logx=True)`

Plotly-based interactive version of `plotRSDs()`

Visualise analytical *versus* biological variance.

Plot RSDs calculated in study-reference samples (analytical variance), versus those calculated in study samples (biological variance). RSDs can be visualised either in absolute terms, or as a ratio to analytical variation (*ratio=True*).

`plotRSDsInteractive()` requires that the dataset have at least two samples with the *PrecisionReference assay role*, if present, RSDs calculated on independent sets of *PrecisionReference* samples will also be plotted.

#### Parameters

- **dataset** (*Dataset*) – Dataset object to plot, the object must have greater than one ‘Study Sample’ and ‘Study-Reference Sample’ defined
- **featureName** (*str*) – featureMetadata column name by which to label features
- **ratio** (*bool*) – If *True* plot the ratio of analytical variance to biological variance instead of raw values
- **logx** (*bool*) – If *True* plot RSDs on a log10 scaled axis

`nPYc.plotting.plotScree(R2, Q2=None, title="", xlabel="", ylabel="", savePath=None, figureFormat='png', dpi=72, figureSize=(11, 7))`

Plot a barchart of variance explained (R2) and predicted (Q2) (if available) for each PCA component.

#### Parameters

- **R2** (*numpy.array*) – PCA R2 values
- **Q2** (*numpy.array*) – PCA Q2 values
- **title** (*str*) – Title for the plot
- **xlabel** (*str*) – Label for the x-axis
- **ylabel** (*str*) – Label for the y-axis

`nPYc.plotting.plotOutliers(values, runOrder, sampleType=None, addViolin=False, Fcrit=None, FcritAlpha=None, PcritPercentile=None, title="", xlabel='Run Order', ylabel="", savePath=None, figureFormat='png', dpi=72, figureSize=(11, 7))`

Plot scatter plot of PCA outlier stats sumT (strong) or DmodX (moderate), with a line at [25, 50, 75, 95, 99] quantiles and at a critical value if specified

#### Parameters

- **values** (*numpy.array*) – dModX or sum of scores, measure of ‘fit’ for each sample
- **runOrder** (*numpy.array*) – Order of sample acquisition (samples are plotted in this order)
- **sampleType** (*pandas.Series*) – Sample type of each sample, must be from ‘Study Sample’, ‘Study Reference’, ‘Long-Term Reference’, or ‘Sample’ (see `multivariateReport.py`)
- **addViolin** (*bool*) – If True adds a violin plot of distribution of values
- **Fcrit** (*float*) – If not none, plots a line at Fcrit
- **FcritAlpha** (*float*) – Alpha value for Fcrit (for legend)
- **PcritPercentile** (*float*) – If not none, plots a line at this quantile
- **title** (*str*) – Title for the plot
- **xlabel** (*str*) – Label for the x-axis

`nPYc.plotting.plotSpectralVariance(dataset, classes=None, quantiles=(25, 75), average='median', xlim=None, **kwargs)`

Plot the average spectral profile of dataset, optionally with the bounds of variance calculated from *quantiles* shaded. By specifying a column from *dataset.sampleMetadata* in the *classes* argument, individual averages and ranges will be plotted for each unique label in *dataset.sampleMetadata[classes]*.

#### Parameters

- **dataset** (*Dataset*) – Data to plot
- **classes** (*None or column in dataset.sampleMetadata*) – Plot by distinct classes specified
- **quantiles** (*None or (min, max)*) – Plot these quantile bounds
- **average** (*str*) – Method to calculate average spectrum, defaults to ‘median’, may also be ‘mean’
- **xlim** (*None or (float, float)*) – Tuple of (min, max) values to scale the x-axis to

- **logy** (*bool*) – If True plot intensities on a log10 scale
- **title** (*str*) – Text to title each plot with

`nPYc.plotting.plotScores` (*pcaModel*, *classes=None*, *classType=None*, *components=None*, *alpha=0.05*, *plotAssociation=None*, *title=""*, *xlabel=""*, *figures=None*, *savePath=None*, *figureFormat='png'*, *dpi=72*, *figureSize=(11, 7)*)

Plot PCA scores for each pair of components in PCAmodel, coloured by values defined in classes, and with Hotelling's T2 ellipse (95%)

#### Parameters

- **pcaModel** (*ChemometricsPCA*) – PCA model object (scikit-learn based)
- **classes** (*pandas.Series*) – Measurement/groupings associated with each sample, e.g., BMI/treatment status
- **classType** (*str*) – Type of data in *classes*, either 'Plot Sample Type', 'categorical' or 'continuous', must be specified if *classes* is not None. If *classType* is 'Plot Sample Type', *classes* expects 'Study Sample', 'Study Reference', 'Long-Term Reference', 'Serial Dilution' or 'Sample'.
- **components** (*tuple (int, int)*) – If None plots all components in model, else plots those specified in components
- **alpha** (*float*) – Significance value for plotting Hotellings ellipse
- **plotAssociation** (*bool*) – If True, plots the association between each set of PCA scores and the metadata values
- **significance** (*numpy.array*) – Significance of association of scores from each component with values in classes from correlation or Kruskal-Wallis test for example (see *multivariateReport.py*)
- **title** (*str*) – Title for the plot
- **xlabel** (*str*) – Label for the x-axis
- **figures** (*dict*) – If not None, saves location of each figure for output in html report (see *multivariateReport.py*)

`nPYc.plotting.plotScoresInteractive` (*dataTrue*, *pcaModel*, *colourBy*, *components=[1, 2]*, *alpha=0.05*, *withExclusions=False*)

Interactively visualise PCA scores (coloured by a given sampleMetadata field, and for a given pair of components) with plotly, provides tooltips to allow identification of samples.

#### Parameters

- **dataTrue** (*Dataset*) – Dataset
- **object pcaModel** (*PCA*) – PCA model object (scikit-learn based)
- **colourBy** (*str*) – **sampleMetadata** field name to of which values to colour samples by
- **components** (*list*) – List of two integers, components to plot
- **alpha** (*float*) – Significance value for plotting Hotellings ellipse
- **withExclusions** (*bool*) – If True, only report on features and samples not masked by the sample and feature masks; must match between data and *pcaModel*

`nPYc.plotting.plotLoadings` (*pcaModel*, *msData*, *title=""*, *figures=None*, *savePath=None*, *figureFormat='png'*, *dpi=72*, *figureSize=(11, 7)*)

Plot PCA loadings for each component in PCAmodel. For NMR data plots the median spectrum coloured by the loading. For MS data plots an ion map (rt vs. mz) coloured by the loading.

#### Parameters

- **pcaModel** (*ChemometricsPCA*) – PCA model object (scikit-learn based)
- **msData** (*Dataset*) – Dataset object
- **title** (*str*) – Title for the plot
- **figures** (*dict*) – If not None, saves location of each figure for output in html report (see `multivariateReport.py`)

`nPYc.plotting.plotLoadingsInteractive` (*dataTrue*, *pcaModel*, *component=1*, *withExclusions=False*)

Interactively visualise PCA loadings (for a given pair of components) with plotly, provides tooltips to allow identification of features.

For MS data, plots RT vs. mz; for NMR plots ppm vs spectral intensity. Plots are coloured by the weight of the loadings.

#### Parameters

- **dataTrue** (*Dataset*) – Dataset
- **pcaModel** (*ChemometricsPCA*) – PCA model object (scikit-learn based)
- **component** (*int*) – Component(s) to plot (one component (int) or list of two integers)
- **withExclusions** (*bool*) – If True, only report on features and samples not masked by the sample and feature masks; must match between data and *pcaModel*

`nPYc.plotting.plotDiscreteLoadings` (*pcaModel*, *nbComponentPerRow=3*, *firstComponent=1*, *sort=True*, *\*\*kwargs*)

Plot loadings for a linear model as a set of parallel vertical scatter plots.

#### Parameters

- **pcaModel** (*ChemometricsPCA*) – Model to plot
- **nbComponentPerRow** (*int*) – Number of side-by-side loading plots to place per row
- **firstComponent** (*int*) – Start plotting components from this component
- **sort** (*bool*) – Plot variable in order of their magnitude in component one

`nPYc.plotting.plotFeatureRanges` (*dataset*, *compounds*, *logx=False*, *histBins=20*, *\*\*kwargs*)

Plot distributions plots of the values listed in **compounds**, on to a set of axes with a linked x-axis.

If reference ranges are specified in *featureMetadata*, a reference range will be drawn behind each plot. If reference ranges are available, distributions that for within the range will be shaded green, and those that fall outside red, where no reference range is available the distribution will be shaded blue.

#### Parameters

- **dataset** (*Dataset*) – Dataset object to plot values from
- **compounds** (*list*) – List of features to plot
- **logx** (*bool*) – Calculate and plot histograms on a log10 scale, if the minumn values is below 1, the histogram is calculated by adding one to all values
- **histBins** (*int*) – Number of bins for histograms



```
nPYc.plotting.plotMetadataDistribution(sampleMetadata, valueType, figures=None,
                                     savePath=None, figureFormat='png', dpi=72,
                                     figureSize=(11, 7))
```

Plot the distribution of a set of data, e.g., `sampleMetadata` fields. Plots a bar chart for categorical data, or a histogram for continuous data.

#### Parameters

- **sampleMetadata** (*dataset.sampleMetadata*) – Set of measurements/groupings associated with each sample, note can contain multiple columns, but they must be of one **valueType**
- **valueType** (*str*) – Type of data contained in **sampleMetadata**, one of `continuous`, `categorical` or `date`
- **figures** (*dict*) – If not `None`, saves location of each figure for output in html report (see `multivariateReport.py`)

```
nPYc.plotting.plotLOQRunOrder(targetedData, addCalibration=True, compareBatch=True, title="",
                              savePath=None, figureFormat='png', dpi=72, figureSize=(11, 7))
```

Visualise ratio of LLOQ and ULOQ by run order, separated by batch. Option to add barchart that summarises across batch

#### Parameters

- **targetedData** (*TargetedDataset*) – TargetedDataset object
- **addCalibration** (*bool*) – If `True` add calibration samples
- **compareBatch** (*bool*) – If `True` add barchart across batch, separated by `SampleType`
- **title** (*str*) – Title for the plot
- **savePath** (*None or str*) – If `None` plot interactively, otherwise save the figure to the path specified
- **figureFormat** (*str*) – If saving the plot, use this format
- **dpi** (*int*) – Plot resolution
- **figureSize** (*tuple(float, float)*) – Dimensions of the figure

#### Raises

- **ValueError** – if `targetedData` does not satisfy to the TargetedDataset definition for QC
- **ValueError** – if `calibration` does not match the number of batch

```
nPYc.plotting.plotFeatureLOQ(tData, splitByBatch=True, plotBatchLOQ=False, zoomLOQ=False,
                             logY=False, tightYLim=True, nbPlotPerRow=3,
                             metabolitesPerPlot=5, withExclusions=True, savePath=None,
                             figureFormat='png', dpi=72, figureSize=(11, 7))
```

Violin plot for each feature with line at LOQ concentrations. Option to split by batch, add each batch LOQs, split by `SampleType`.

#### Parameters

- **tData** (*TargetedDataset*) – TargetedDataset
- **splitByBatch** (*bool*) – If `True` separate each violin plot by batch
- **plotBatchLOQ** (*bool*) – If `True` add lines at LOQs (LLOQ/ULOQ) for each batch, and points for samples that will be out of LOQ

- **zoomLOQ** (*bool*) – If `True` plots a zoomed ULOQ plot on top, all data in the centre and a zoomed LLOQ plot at the bottom
- **logY** (*bool*) – If `True` log-scale the y-axis
- **tightYLim** (*bool*) – if `True` ylim are close to the points but can let LOQ lines outside, if `False` LOQ lines will be part of the plot.
- **nbPlotPerRow** (*int*) – Number of plots to place on each row
- **metabolitesPerPlot** (*int*) – Maximum number of metabolites to plot in on single figure
- **savePath** (*None or str*) – If `None` plot interactively, otherwise save the figure to the path specified
- **figureFormat** (*str*) – If saving the plot, use this format
- **dpi** (*int*) – Plot resolution
- **figureSize** (*tuple(float, float)*) – Dimensions of the figure

**Raises ValueError** – if `targetedData` does not satisfy to the `TargetedDataset` definition for QC

`nPYc.plotting.plotVariableScatter` (*inputTable*, *logX=False*, *xLim=None*, *xLabel=""*, *yLabel=""*, *sampletypeColor=False*, *hLines=None*, *hLineStyle='-'*, *hBox=None*, *vLines=None*, *vLineStyle=':'*, *vBox=None*, *savePath=None*, *figureFormat='png'*, *dpi=72*, *figureSize=(11, 7)*)

Plot values on x-axis, with ordering on the y-axis. Entries as rows are placed on the x-axis, values of all columns are plotted on y-axis with different colors. If `sampletypeColor=True`, only columns named as `SampleTypes` will be plotted and colored according to other reports, otherwise all columns are plotted. Ordering of the rows is conserved, the first item is placed at the top of the y-axis and the last row is at the bottom. If a column [`'yName'`] is present, it is employed to label each y-axis entry.

#### Parameters

- **inputTable** (*dataframe*) – `DataFrame` or accuracy or precision values, with features as rows and sample types as columns ([`'Study Sample'`, `'Study Pool'`, `'External Reference'`, `'All Samples'`, `'nan'`]). A `'yName'` column can be present to display the feature name.
- **logX** (*bool*) – If `True` plot values on a log10 scaled x axis
- **xLim** (*None or tuple(float, float)*) – Tuple of (min, max) values to plot
- **xLabel** (*str*) – X-axis label
- **yLabel** (*str*) – Y-axis label
- **sampletypeColor** (*bool*) – If `True` only the `sampleType` columns are plotted with colors matching other reports
- **hLines** (*None or list*) – `None` or list of y positions at which to plot an horizontal line. Features are positioned from 1 to `nFeat`
- **hLineStyle** (*str*) – One of the axhline linestyle (`'-'`, `'--'`, `'-.'`, `':'`)
- **hBox** (*None or list*) – `None` or list of tuple of y positions defining horizontal box. Features are positioned from 1 to `nFeat`
- **vLines** (*None or list*) – `None` or list of v positions at which to plot an vertical line. Unit is the same as the v axis.
- **vLineStyle** (*str*) – One of the axvline linestyle (`'-'`, `'--'`, `'-.'`, `':'`)

- **vBox** (*None or list*) – None or list of tuple of x positions defining horizontal box. Features are positioned from 1 to nFeat
- **color** (*None or seaborn.palettes.\_ColorPalette*) – Allows the default colour pallet to be overridden
- **savePath** (*None or str*) – If None plot interactively, otherwise save the figure to the path specified

```
nPYc.plotting.plotAccuracyPrecision(tData, accuracy=True, percentRange=None,
                                   savePath=None, figureFormat='png', dpi=72, figureSize=(11, 7))
```

Plot Accuracy or Precision for a TargetedDataset.

Features at all present concentrations are shown on the y-axis, with accuracy or precision values on the x-axis. Accuracy are centered around 100%. If Precision values cover too wide a range, x-axis is log transformed.

#### Parameters

- **tData** (*TargetedDataset*) – TargetedDataset object to plot
- **accuracy** (*bool*) – If True plot the Accuracy of each measurements, if False plot the Precision of measurements.
- **percentRange** (*None or float*) – If float [0, inf], add a rectangle covering the range of acceptable percentage; for Accuracy 100 +/- percentage, for Precision 0 - percentage.
- **savePath** (*None or str*) – If None plot interactively, otherwise save the figure to the path specified
- **figureFormat** (*str*) – If saving the plot, use this format
- **dpi** (*int*) – Plot resolution
- **figureSize** (*tuple(float, float)*) – Dimensions of the figure

#### Raises

- **ValueError** – if targetedData does not satisfy to the TargetedDataset definition for QC
- **ValueError** – if percentRange is not 'None' or float

```
nPYc.plotting.plotCalibrationInteractive(nmrData)
```

Build Plotly figure of calibration

**Parameters** **nmrData** (*NMRDataset*) – Dataset to visualise

**Returns** Plotly figure object for display with iplot()

**Return type** plotly.graph\_objs.Figure

```
nPYc.plotting.plotLineWidth(nmrData, **kwargs)
```

Visualise spectral line widths, plotting the median spectrum, the 95% variance, and any spectra where line width can not be calculated or exceeds the cutoff specified in `nmrData.Attributes['LWpeakRange']`.

#### Parameters

- **nmrData** (*NMRDataset*) – Dataset object
- **savePath** (*None or str*) – If None, plot interactively, otherwise attempt to save at this path

```
nPYc.plotting.plotLineWidthInteractive(nmrData)
```

Interactive Plotly version of `py:func:plotLineWidth`

Visualise spectral line widths, plotting the median spectrum, the 95% variance, and any spectra where line width can not be calculated or exceeds the cutoff specified in `nmrData.Attributes['LWpeakRange']`.

#### Parameters

- **nmrData** (`NMRDataset`) – Dataset object
- **savePath** (*None or str*) – If *None*, plot interactively, otherwise attempt to save at this path

`nPYc.plotting.plotBaseline(nmrData, savePath=None, **kwargs)`

Plot spectral baseline at the high and low end of the spectrum. Visualise the median, bounds of 95% variance, and outliers.

#### Parameters

- **nmrData** (`NMRDataset`) – Dataset object
- **savePath** (*None or str*) – If *None*, plot interactively, otherwise attempt to save at this path

`nPYc.plotting.plotBaselineInteractive(nmrData)`

Interactive Plotly version of `py:func:plotBaseline`.

Plot spectral baseline at the high and low end of the spectrum. Visualise the median, bounds of 95% variance, and outliers.

**Parameters** **nmrData** (`NMRDataset`) – Dataset object

`nPYc.plotting.plotSolventResonance(nmrData, **kwargs)`

Plot the solvent region to be cut from the spectrum along with spectra failing solvent region checks.

#### Parameters

- **nmrData** (`NMRDataset`) – Dataset to plot
- **savePath** (*None or str*) – If *None* draw interactively, otherwise save to this path

`nPYc.plotting.plotSolventResonanceInteractive(nmrData, title='Residual solvent resonance')`

Plotly interactive version of `plotSolventResonance()`

Plot the solvent region to be cut from the spectrum along with spectra failing solvent region checks.

**Parameters** **nmrData** (`NMRDataset`) – Dataset to plot

**Returns** Plotly figure object to plot with `iPlot`

`nPYc.plotting.plotSpectraInteractive(dataset, samples=None, xlim=None, featureNames=None, sampleLabels='Sample ID', nmrDataset=True)`

Plot spectra from *dataset*.

#:param Dataset dataset: Dataset to plot from :param samples: Index of samples to plot, if *None* plot all spectra :type samples: *None* or list of int :param xlim: Tuple of (minimum value, maximum value) defining a feature range to plot :type xlim: (float, float)

`nPYc.plotting.plotIonMapInteractive(dataset, title=None, xlim=None, ylim=None, logx=False, logy=False, featureName='Feature Name')`

Visualise features in a `MSDataset`, as an ion map.

Plotting requires the presence of ‘m/z’ and ‘Retention Time’ columns in the *featureMetadata* table.

**Parameters** **msData** (`MSDataset`) – Dataset object to visualise

`nPYc.plotting.plotSpectralVarianceInteractive`(*dataset*, *classes=None*, *quantiles=(25, 75)*, *average='mean'*, *xlim=None*, *title=None*)

Plot the average spectral profile of dataset, optionally with the bounds of variance calculated from *quantiles* shaded. By specifying a column from *dataset.sampleMetadata* in the *classes* argument, individual averages and ranges will be plotted for each unique label in *dataset.sampleMetadata[classes]*.

#### Parameters

- **dataset** (*Dataset*) – Data to plot
- **classes** (*None or column in dataset.sampleMetadata*) – Plot by distinct classes specified
- **quantiles** (*None or (min, max)*) – Plot these quantile bounds
- **average** (*str*) – Method to calculate average spectrum, defaults to ‘median’, may also be ‘mean’
- **xlim** (*None or (float, float)*) – Tuple of (min, max) values to scale the x-axis to

`nPYc.plotting.correlationSpectroscopyInteractive`(*dataset*, *target*, *mode='SHY'*, *correlationMethod='Pearson'*)

Conduct correlation spectroscopy analyses against the samples in *dataset*.

Mode may be one of: - **SHY** Correlate features in *dataset* to values in *target*

#### Parameters

- **dataset** (*Dataset*) – Correlations will be projected into this dataset
- **target** (*numpy.array*) – Correlations are calculated to this
- **mode** (*str*) – Type of analysis to conduct
- **correlationMethod** (*str*) – Type of correlation to calculate, may be ‘Pearson’, or ‘Spearman’

**Returns** Plotly figure

#### Return type

`nPYc.plotting.plotTargetedFeatureDistribution`(*datasetOriginal*, *featureName='Feature Name'*, *featureMask=None*, *sampleTypes=['SS', 'SP', 'ER']*, *logx=False*, *figures=None*, *savePath=None*, *figureFormat='png'*, *dpi=72*, *figureSize=(11, 7)*)

Plot the distribution (violin plots) of a set of features, e.g., peakPantheR outputs, coloured by sample type

#### Parameters

- **dataset** (*MSDataset*) – MSDataset
- **logx** (*bool*) – If True log-scale the x-axis
- **figures** (*dict*) – If not None, saves location of each figure for output in html report (see `_generateMSReport.py`)



Examples of the outputs from *Plotting Functions* implemented in the toolbox.

Fig. 1: *histogram()* - Draw a histogram, optionally segmented by a second parameter.

Fig. 2: *plotBatchAndROCorrection()* - Visualise the run-order and batch correction applied to a dataset.

Fig. 3: *plotTIC()* - Visualise TIC for all or a subset of features in an *MSDataset*, coloured by class, dilution value, or detector voltage.

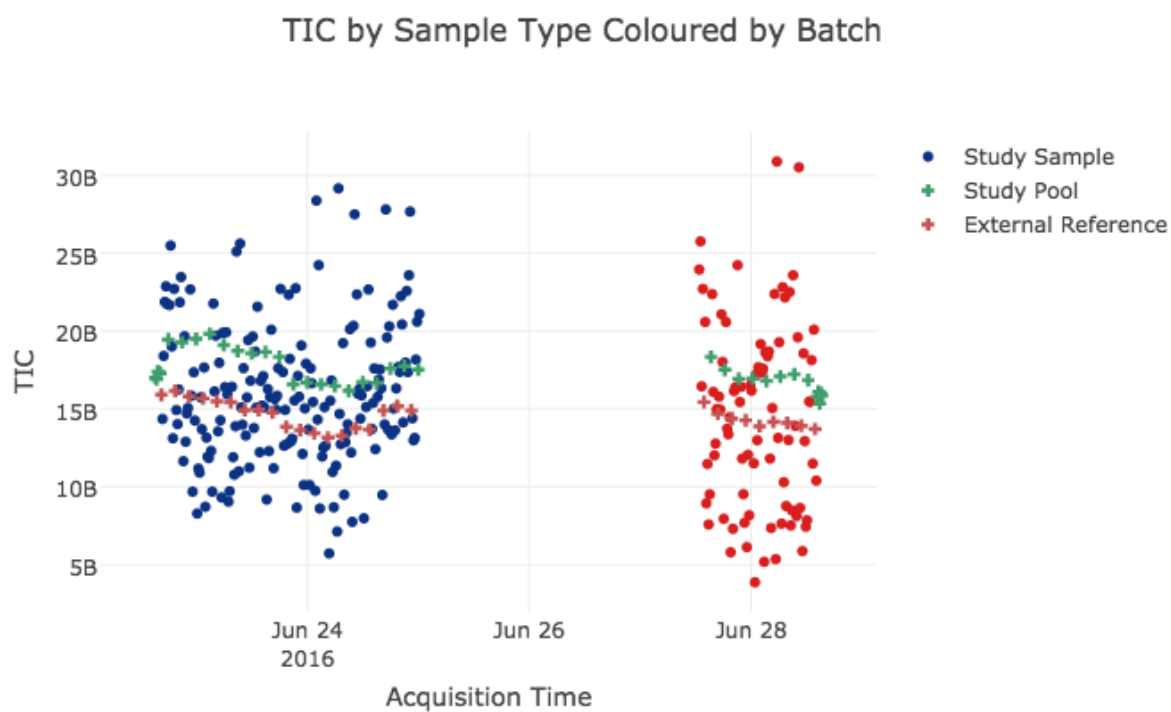


Fig. 4: `plotTICinteractive()` - Interactively visualise TIC vs. run-order for features in an *MSDataset*, coloured by sample type.



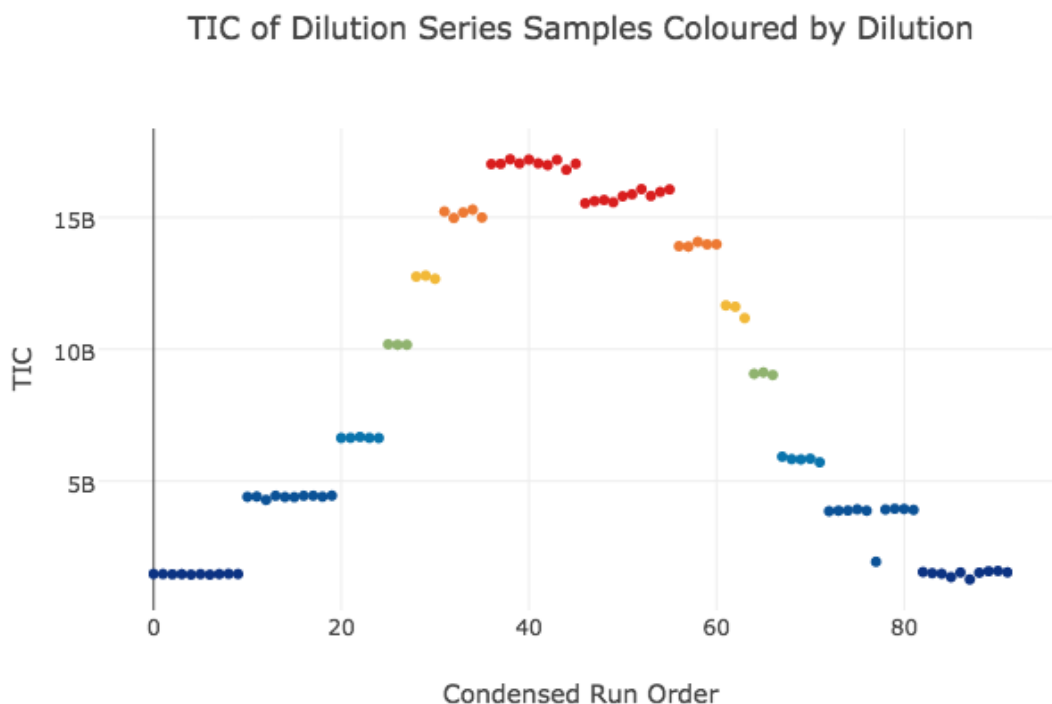


Fig. 5: `plotTICinteractive()` - Interactively visualise TIC vs. run-order of linearity reference samples from an `MSDataset`, coloured by dilution value.

Fig. 6: `plotLRTIC()` - Visualise TIC vs. run-order of linearity reference samples from an `MSDataset`, coloured by dilution value.

Fig. 7: `jointplotRSDvCorrelation()` - Visualise 2D histogram of feature RSDs vs. correlations to dilution, with marginal histograms from `Spectral` datasets.

Fig. 8: `plotIonMap()` - Visualise the features present in an `MSDataset` object in terms of the original analytics. Also has a plotly-based interactive version `plotIonMapInteractive()`.

Fig. 9: `plotRSDs()` - Visualise the analytical and biological variance in *Discretely* sampled datasets.

Fig. 10: `plotScree()` - Plot a barchart of variance explained (R2) and predicted (Q2) (if available) for each PCA component derived from a PCA model generated on `Dataset` datasets.

Fig. 11: `plotScores()` - Plot PCA scores for each pair of components in `PCAModel`, coloured by values defined in classes, and with Hotelling's T2 ellipse (95%), derived from a PCA model generated on `Dataset` datasets.

Fig. 12: `plotOutliers()` - Plot scatter plot of PCA outlier stats sumT (strong) or DmodX (moderate), with a line at [25, 50, 75, 95, 99] quantiles, derived from a PCA model generated on `Dataset` datasets.

Fig. 13: `plotLoadings()` - Plot PCA loadings for each component in PCAmodel. For *NMRDataset* datasets plots the median spectrum coloured by the loading. For *MSDataset* datasets plots an ion map (rt vs. mz) coloured by the loading.

Fig. 14: `plotSpectralVariance()` - Plot of median profile with variance across all samples visualised in *Spectral* datasets. Also has a plotly-based interactive version `plotSpectralVarianceInteractive()`.

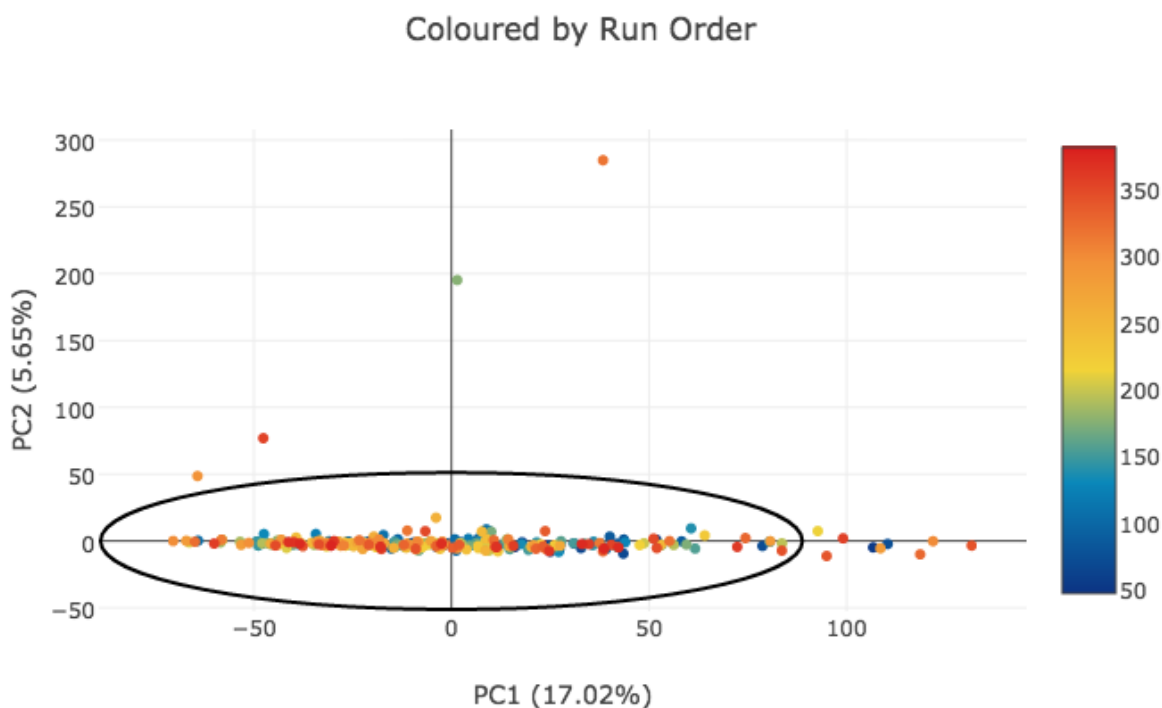


Fig. 15: `plotScoresInteractive()` - Interactively visualise PCA scores (coloured by a given sampleMetadata field, and for a given pair of components) with plotly, provides tooltips to allow identification of samples, derived from a PCA model generated on *Dataset* datasets.

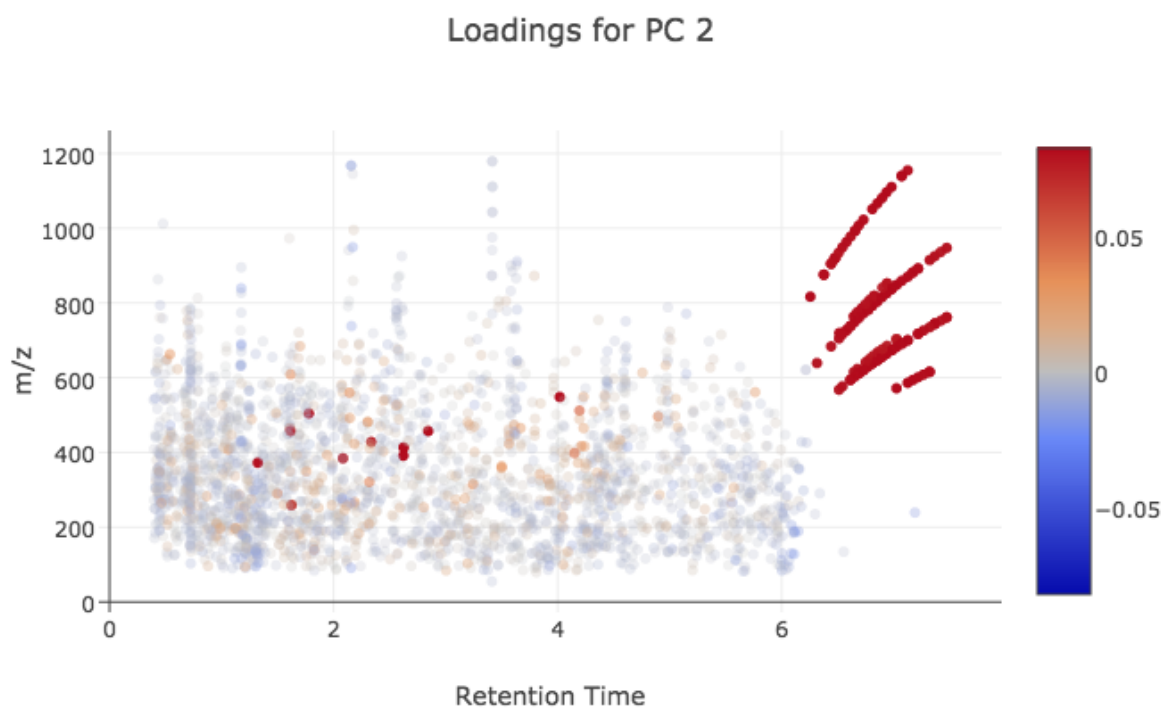


Fig. 16: `plotLoadingsInteractive()` - Interactively visualise PCA loadings (for a given pair of components) with plotly, provides tooltips to allow identification of features., derived from a PCA model generated on *Dataset* datasets.

Fig. 17: `plotDiscreteLoadings()` - Visualise loadings of a ChemometricsPCA model.



**Aliquot** Aliquots are one or more sub-fractions of a *sample* that may be considered functionally equivalent. Setting aside handling considerations, aliquots may be combined or split with no impact on sample composition or the expected result of an *assay*.

**Analytical Batch** Set of *study* and *reference samples* acquired in a single continuous analytical run, without planned interruption *i.e.* instrument maintenance.

**Assay** Analytical procedure, encompassing sample preparation, data-acquisition, and feature extraction, for the characterisation of the chemical composition of samples. The datasets generated by an assay may provide measures as relative or absolute quantifications, for either absolute chemical names, or annotated and unknown *features*.

**Assay Role** The rationale for acquisition of a specific sample (see *AssayRole*).

**Batch Effects** Analytical and preparative influences that may cause a systematic difference in measurements taken at different points in time.

#### Continuum Data

**Spectral Data** Analytical data in which the adjacency of variables is significant. Examples include *NMR* spectra, or mass-spectra recorded in continuum mode.

**Correction Batch** In the ideal case, analytical *batch* and *run-order* effects are detected and corrected based on the *analytical batches* into which the study has been divided. However in the event of unplanned interruptions to an analysis, it may be necessary to further sub-divide the run into a series of correction batches.

**Correlation to Dilution** The Correlation to Dilution provides a measurement of analytical accuracy, expressed as a value between -1 and 1, where the closer the value to 1 the more accurately the feature is measured with respect to its expected concentration. The Correlation to Dilution for *feature x*, is the Pearson correlation coefficient between the feature's measured concentration, and the expected concentration of the sample, calculated from the *Serial Dilution Sample* set.

**Discrete Data** Analytical data in which the adjacency of variables is unimportant to their interpretation. Peak-picked *UPLC-MS*, targeted, and clinical measures are typically of this type.

**Feature** Measured entity from a specific *assay*, that proxies the abundance of a chemical in the assayed sample. Each chemical in a sample may give rise to none, one, or several features in the dataset generated from a specific assay.

### Long-Term Reference

**LTR** A specific *sample type/assay role* combination consisting of samples with *External Reference* and *Precision Reference* assignment. These represent a type of QC sample useful, for example, for between-study comparisons.

**Mass Accuracy** The precision by which the *m/z* of an ion can be measured in *mass spectrometry*. Typically expressed in *ppm* and calculated by:  $\Delta m_i = \frac{(m_i - m_a)}{m_a} \times 10^6$  where  $m_i$  is the observed mass and  $m_a$  is the true mass.

### Mass Spectrometry

**MS** Analytical technology that *assays* a sample in terms of the observed *mass-to-charge ratio* of the constituent compounds.

### Mass-to-Charge Ratio

*m/z* *Mass Spectrometry* term describing the measurement of an ions mass relative to its charge.

**Matrix** The source of a *specimen*, for example, urine, blood-plasma, or serum.

**Notation conventions (code)** Matrices are set *UPPERCASE*, vectors *lowercase*, and scalar values *italic*.

### Nuclear Magnetic Resonance Spectroscopy

**NMR** Analytical technology for *assaying* samples by detection the resonance of atomic nuclei in a magnetic field.

### Participant

#### Subject

**Sample Source** The source of a *study sample* (generated at a *sampling event*), which could represent an individual, experimental site or condition, or other.

**ppm (MS)** Parts-per-Million, used as a measure of *mass accuracy* in *mass spectrometry*.

**ppm (NMR)** Parts-per-Million, a measurement of the chemical shift of a nucleus ( $\nu$ ) in *NMR*, expressed as a ratio to the spectrometer frequency ( $\nu_{\text{ref}}$ ) by:  $\delta = \frac{\nu - \nu_{\text{ref}}}{\nu_{\text{ref}}}$ .

**Preparative Batch** A group of one or more *sample batches* handled and prepared together, using a single batch of reagents.

**Reference Sample** Reference samples are measured to characterise the stability of assays during the course of an acquisition, and account for platform dependent analytical variability. There are several common forms of reference sample, including *Study Reference*, *Long-Term Reference*.

### Relative Standard Deviation

**RSD** The RSD provides a measurement of analytical precision, expressed as a percentage. The RSD is calculated for *feature x*, from repeated measurements (typically of the *study reference* samples), by:  $rsd(x) = \frac{\sigma_x}{\mu_x} \times 100$ .

**Repeat Assay** Replicate *analytical data* acquired from a sample that augments any data previously acquired. For example an interruption in the acquisition of an MS batch may cause an additional dilution series to be acquired when analysis resumes.

**Rerun Assay** Replicate *analytical data* acquired from a sample that obsoletes any data previously acquired. For example, *study samples* reacquired following analytical issues are reruns.

**Resolution** The ability of an instrument to separate two signals.

In *NMR* resolution is directly related to the magnetic field strength, and typically expressed in terms of the resonant frequency of the hydrogen nuclei in H<sub>2</sub>O at room temperature.

In *MS* resolution is measured and calculated by  $r = \frac{m_i}{w_{1/2}}$ , where  $m_i$  is the nominal mass of an ion, and  $w_{1/2}$  is the measured peak-width at half-height.

**Retention Time** Measurement of the time of elution of a feature as observed in a specific *UPLC-MS* chromatographic method. Internally, all nPYc toolbox retention times are expressed in seconds unless otherwise noted.

**Run Order** The sequence in which *samples* are *assayed*.

**Run-Order Effects** Analytical factors that may affect the measurement of *features* in a dataset by introducing progressive assay-to-assay biases in measurement. Examples include the gradual decline in observed intensity of measurement in ToF MS detectors.

**Sample** A single specimen to be *assayed*. May be divided into two *broad classes*, *study samples* which form the core of an analysis, and *reference samples*, that allow that characterisation of analytical performance.

**Sample Assay** Analytical data acquired by a single *assay*, from a single physical specimen.

**Sample Base Name** Common name for all comparable assays of the same sample. For example, reacquisitions of the same sample will share an identical Base Name.

**Sample Batch** A collection of *study samples* (typically 80, to allow formatting onto a 96-well plate with room for *reference samples*) plus some number of *reference samples*, prepared and analysed together.

**Sample File Name** Unique name of an assay data file. Two *sample assays* acquired from the sample physical sample (for example, a *rerun*), will have unique Sample File Names.

**Sample Type** The overarching compositional class of a specific sample (see *SampleType*).

**Sampling Event** The specific point in time at which a *sample* was generated. One sampling event may produce several equivalent *aliquots*. Note that obtaining samples of blood-plasma and urine from a *participant* at the same time is considered two sampling events, as the biofluids obtained are not interchangeable.

### Serial Dilution Sample

**SRDS** A specific *sample type/assay role* combination consisting of samples with *Study Pool* and *Linearity Reference* assignment. Serial Dilution Samples consist of a number of pooled QC samples diluted to known concentrations and acquired to assess the linearity of response of *features* during analysis.

**Study** A collection of *samples* for analysis, constituting a single project.

### Study Reference

**SR** A specific *sample type/assay role* combination consisting of samples with *Study Sample* and *Precision Reference* assignment. These represent the classic QC sample used in profiling studies to assess analytical stability.

### Study Sample

**SS** Samples comprising the *study*.

### Ultra-Performance Liquid Chromatography Mass-Spectrometry

**UPLC-MS** Analytical technology for *assaying* samples, coupling chromatographic separation with *mass detection*.

### Units

Where unspecified units used in the nPYc toolbox are as follows:

Variable	Unit	Datatype	Interpretation
Sample inclusion		bool	True == included, False == excluded
Feature inclusion		bool	True == included, False == excluded
<i>Run order</i>		int	Ascending rank order
Times & Dates		datetime	Export / import as <b>RFC 3339</b>
Fluid volumes	Milliliters (ml)	float	
Ionisation Mode		<i>Polarity</i>	
Ionisation Type		<i>Ionisation</i>	
<i>Retention Time</i>	Seconds (s)	float	
Atomic Mass	Unified atomic mass units (u)	float	
<i>NMR Chemical Shift</i>	<i>PPM</i>	float	
Collision Energy	Volts (v)	float	

- `genindex`
- `modindex`
- `search`



### n

- nPYc, [??](#)
- nPYc.batchAndROCorrection, [64](#)
- nPYc.enumerations, [87](#)
- nPYc.multivariate.exploratoryAnalysisPCA,  
[68](#)
- nPYc.plotting, [93](#)
- nPYc.utilities, [91](#)
- nPYc.utilities.extractParams, [47](#)
- nPYc.utilities.normalisation, [73](#)



## Symbols

`_generateReportMS` (class in *nPYc.reports*), 60  
`_generateReportNMR` (class in *nPYc.reports*), 61  
`_generateReportTargeted` (class in *nPYc.reports*), 61  
`_generateSampleReport` (class in *nPYc.reports*), 54

## A

`accuracyPrecision()` (*nPYc.objects.TargetedDataset* method), 43  
`addFeatureInfo()` (*nPYc.objects.Dataset* method), 25  
`addSampleInfo()` (*nPYc.objects.Dataset* method), 25  
`addSampleInfo()` (*nPYc.objects.MSDataset* method), 29  
`addSampleInfo()` (*nPYc.objects.NMRDataset* method), 33  
`addSampleInfo()` (*nPYc.objects.TargetedDataset* method), 42  
`Aliquot`, 113  
`amendBatches()` (*nPYc.objects.MSDataset* method), 30  
`Analytical Batch`, 113  
`AnalyticalPlatform` (class in *nPYc.enumerations*), 89  
`AnalyticalPlatform` (*nPYc.objects.Dataset* attribute), 21  
`applyMasks()` (*nPYc.objects.Dataset* method), 25  
`applyMasks()` (*nPYc.objects.MSDataset* method), 28  
`applyMasks()` (*nPYc.objects.TargetedDataset* method), 41  
`artifactualFilter()` (*nPYc.objects.MSDataset* method), 30  
`artifactualLinkageMatrix` (*nPYc.objects.MSDataset* attribute), 28  
`Assay`, 113

`Assay` (*nPYc.enumerations.AssayRole* attribute), 88  
`Assay Role`, 113  
`AssayRole` (class in *nPYc.enumerations*), 88  
`Attributes` (*nPYc.objects.Dataset* attribute), 21

## B

`backcalculatedIS` (*nPYc.enumerations.CalibrationMethod* attribute), 89  
`Batch Effects`, 113  
`Blank` (*nPYc.enumerations.AssayRole* attribute), 88  
`BrukerivDrEstimate` (*nPYc.enumerations.QuantificationType* attribute), 89  
`BrukerivDrQuant` (*nPYc.enumerations.QuantificationType* attribute), 89  
`buildFileList()` (in module *nPYc.utilities*), 91

## C

`CalibrationMethod` (class in *nPYc.enumerations*), 89  
`Continuum` (*nPYc.enumerations.VariableType* attribute), 87  
`Continuum Data`, 113  
`Correction Batch`, 113  
`correctMSdataset()` (in module *nPYc.batchAndROCorrection*), 64  
`Correlation to Dilution`, 113  
`correlationSpectroscopyInteractive()` (in module *nPYc.plotting*), 105  
`correlationToDilution` (*nPYc.objects.MSDataset* attribute), 28

## D

`Dataset` (class in *nPYc.objects*), 20  
`DatasetLevel` (class in *nPYc.enumerations*), 88  
`DESI` (*nPYc.enumerations.Ionisation* attribute), 87  
`Discrete` (*nPYc.enumerations.VariableType* attribute), 87  
`Discrete Data`, 113

## E

EI (*nPYc.enumerations.Ionisation attribute*), 87  
Empty (*nPYc.enumerations.DatasetLevel attribute*), 88  
ESI (*nPYc.enumerations.Ionisation attribute*), 87  
excludeFeatures() (*nPYc.objects.Dataset method*), 26  
excludeFeatures() (*nPYc.objects.MSDataset method*), 30  
excludeSamples() (*nPYc.objects.Dataset method*), 26  
exploratoryAnalysisPCA() (*in module nPYc.multivariate.exploratoryAnalysisPCA*), 68  
exportDataset() (*nPYc.objects.Dataset method*), 26  
exportDataset() (*nPYc.objects.TargetedDataset method*), 38  
ExternalReference (*nPYc.enumerations.SampleType attribute*), 88  
extractParams() (*in module nPYc.utilities.extractParams*), 47

## F

Feature, 113  
featureMask (*nPYc.objects.Dataset attribute*), 21  
featureMetadata (*nPYc.objects.Dataset attribute*), 20

## G

getFeatures() (*nPYc.objects.Dataset method*), 27

## H

histogram() (*in module nPYc.plotting*), 93

## I

initialiseMasks() (*nPYc.objects.Dataset method*), 24  
initialiseMasks() (*nPYc.objects.MSDataset method*), 30  
intensityData (*nPYc.objects.Dataset attribute*), 22  
Ionisation (*class in nPYc.enumerations*), 87  
IS (*nPYc.enumerations.QuantificationType attribute*), 89

## J

jointplotRSDvCorrelation() (*in module nPYc.plotting*), 95

## L

LinearityReference (*nPYc.enumerations.AssayRole attribute*), 88  
log (*nPYc.objects.Dataset attribute*), 22  
Long-Term Reference, 114

LTR, 114

## M

m/z, 114  
MALDI (*nPYc.enumerations.Ionisation attribute*), 87  
Mass Accuracy, 114  
Mass Spectrometry, 114  
Mass-to-Charge Ratio, 114  
Matrix, 114  
mergeLimitsOfQuantification() (*nPYc.objects.TargetedDataset method*), 38  
MethodReference (*nPYc.enumerations.SampleType attribute*), 88  
Monitored (*nPYc.enumerations.QuantificationType attribute*), 89  
MS, 114  
MS (*nPYc.enumerations.AnalyticalPlatform attribute*), 89  
MSDataset (*class in nPYc.objects*), 27  
multivariateReport (*class in nPYc.reports*), 71

## N

name (*nPYc.objects.Dataset attribute*), 22  
Negative (*nPYc.enumerations.Polarity attribute*), 88  
NMR, 114  
NMR (*nPYc.enumerations.AnalyticalPlatform attribute*), 89  
nmrCalibration (*nPYc.enumerations.CalibrationMethod attribute*), 89  
NMRDataset (*class in nPYc.objects*), 33  
noCalibration (*nPYc.enumerations.CalibrationMethod attribute*), 89  
noFeatures (*nPYc.objects.Dataset attribute*), 22  
noIS (*nPYc.enumerations.CalibrationMethod attribute*), 89  
Normalisation (*nPYc.objects.Dataset attribute*), 22  
normalisation\_coefficients (*nPYc.utilities.normalisation.NullNormaliser attribute*), 74  
normalisation\_coefficients (*nPYc.utilities.normalisation.ProbabilisticQuotientNormaliser attribute*), 74  
normalisation\_coefficients (*nPYc.utilities.normalisation.TotalAreaNormaliser attribute*), 75  
normalise() (*nPYc.utilities.normalisation.NullNormaliser method*), 74  
normalise() (*nPYc.utilities.normalisation.ProbabilisticQuotientNormaliser method*), 74  
normalise() (*nPYc.utilities.normalisation.TotalAreaNormaliser method*), 75  
noSamples (*nPYc.objects.Dataset attribute*), 22  
Notation conventions (code), 114  
nPYc (module), 1

- nPYc.batchAndROCorrection (*module*), 64  
 nPYc.enumerations (*module*), 87  
 nPYc.multivariate.exploratoryAnalysisPCA (*module*), 68  
 nPYc.plotting (*module*), 93  
 nPYc.utilities (*module*), 91  
 nPYc.utilities.extractParams (*module*), 47  
 nPYc.utilities.normalisation (*module*), 73  
 Nuclear Magnetic Resonance Spectroscopy, 114  
 NullNormaliser (*class in nPYc.utilities.normalisation*), 74
- ## O
- Other (*nPYc.enumerations.AnalyticalPlatform attribute*), 89  
 otherCalibration (*nPYc.enumerations.CalibrationMethod attribute*), 89
- ## P
- Participant, 114  
 plot () (*nPYc.objects.NMRDataset method*), 34  
 plotAccuracyPrecision () (*in module nPYc.plotting*), 103  
 plotBaseline () (*in module nPYc.plotting*), 104  
 plotBaselineInteractive () (*in module nPYc.plotting*), 104  
 plotBatchAndROCorrection () (*in module nPYc.plotting*), 94  
 plotCalibrationInteractive () (*in module nPYc.plotting*), 103  
 plotCorrelationToLRbyFeature () (*in module nPYc.plotting*), 96  
 plotDiscreteLoadings () (*in module nPYc.plotting*), 100  
 plotFeatureLOQ () (*in module nPYc.plotting*), 101  
 plotFeatureRanges () (*in module nPYc.plotting*), 100  
 plotIonMap () (*in module nPYc.plotting*), 96  
 plotIonMapInteractive () (*in module nPYc.plotting*), 104  
 plotLineWidth () (*in module nPYc.plotting*), 103  
 plotLineWidthInteractive () (*in module nPYc.plotting*), 103  
 plotLoadings () (*in module nPYc.plotting*), 99  
 plotLoadingsInteractive () (*in module nPYc.plotting*), 100  
 plotLOQRunOrder () (*in module nPYc.plotting*), 101  
 plotLRTIC () (*in module nPYc.plotting*), 95  
 plotMetadataDistribution () (*in module nPYc.plotting*), 100  
 plotOutliers () (*in module nPYc.plotting*), 98  
 plotRSDs () (*in module nPYc.plotting*), 96  
 plotRSDsInteractive () (*in module nPYc.plotting*), 97  
 plotScores () (*in module nPYc.plotting*), 99  
 plotScoresInteractive () (*in module nPYc.plotting*), 99  
 plotScree () (*in module nPYc.plotting*), 97  
 plotSolventResonance () (*in module nPYc.plotting*), 104  
 plotSolventResonanceInteractive () (*in module nPYc.plotting*), 104  
 plotSpectraInteractive () (*in module nPYc.plotting*), 104  
 plotSpectralVariance () (*in module nPYc.plotting*), 98  
 plotSpectralVarianceInteractive () (*in module nPYc.plotting*), 104  
 plotTargetedFeatureDistribution () (*in module nPYc.plotting*), 105  
 plotTIC () (*in module nPYc.plotting*), 94  
 plotTICInteractive () (*in module nPYc.plotting*), 95  
 plotVariableScatter () (*in module nPYc.plotting*), 102  
 Polarity (*class in nPYc.enumerations*), 88  
 Positive (*nPYc.enumerations.Polarity attribute*), 88  
 ppm (*MS*), 114  
 ppm (*NMR*), 114  
 PrecisionReference (*nPYc.enumerations.AssayRole attribute*), 88  
 Preparative Batch, 114  
 ProbabilisticQuotientNormaliser (*class in nPYc.utilities.normalisation*), 74  
 ProceduralBlank (*nPYc.enumerations.SampleType attribute*), 88  
 Profile (*nPYc.enumerations.VariableType attribute*), 87
- ## Q
- QCReady (*nPYc.enumerations.DatasetLevel attribute*), 88  
 QuantAltLabeledAnalogue (*nPYc.enumerations.QuantificationType attribute*), 89  
 QuantificationType (*class in nPYc.enumerations*), 88  
 QuantOther (*nPYc.enumerations.QuantificationType attribute*), 89  
 QuantOwnLabeledAnalogue (*nPYc.enumerations.QuantificationType attribute*), 89
- ## R
- reference (*nPYc.utilities.normalisation.ProbabilisticQuotientNormaliser*

attribute), 74  
 Reference Sample, 114  
 Relative Standard Deviation, 114  
 Repeat Assay, 114  
 Rerun Assay, 114  
 Resolution, 114  
 Retention Time, 114  
 RFC  
     RFC 3339, 116  
 RSD, 114  
 rsd() (in module nPYc.utilities), 91  
 rsdsBySampleType() (in module nPYc.utilities), 91  
 rsdSP (nPYc.objects.MSDataset attribute), 28  
 rsdSP (nPYc.objects.TargetedDataset attribute), 38  
 rsdSS (nPYc.objects.MSDataset attribute), 28  
 rsdSS (nPYc.objects.TargetedDataset attribute), 38  
 Run Order, 115  
 Run-Order Effects, 115

## S

Sample, 115  
 Sample Assay, 115  
 Sample Base Name, 115  
 Sample Batch, 115  
 Sample File Name, 115  
 Sample Source, 114  
 Sample Type, 115  
 sampleMask (nPYc.objects.Dataset attribute), 21  
 sampleMetadata (nPYc.objects.Dataset attribute), 20  
 SampleType (class in nPYc.enumerations), 87  
 Sampling Event, 115  
 saveFeatureMask() (nPYc.objects.MSDataset method), 29  
 sequentialPrecision() (in module nPYc.utilities), 91  
 Serial Dilution Sample, 115  
 Spectral (nPYc.enumerations.VariableType attribute), 87  
 Spectral Data, 113  
 SR, 115  
 SRDS, 115  
 SS, 115  
 Study, 115  
 Study Reference, 115  
 Study Sample, 115  
 StudyPool (nPYc.enumerations.SampleType attribute), 88  
 StudySample (nPYc.enumerations.SampleType attribute), 88  
 Subject, 114

## T

TargetedDataset (class in nPYc.objects), 34

TotalAreaNormaliser (class in nPYc.utilities.normalisation), 75

## U

Ultra-Performance Liquid  
     Chromatography  
     Mass-Spectrometry, 115  
 Unknown (nPYc.enumerations.DatasetLevel attribute), 88  
 updateMasks() (nPYc.objects.Dataset method), 24  
 updateMasks() (nPYc.objects.MSDataset method), 28  
 updateMasks() (nPYc.objects.NMRDataset method), 34  
 updateMasks() (nPYc.objects.TargetedDataset method), 41  
 UPLC-MS, 115

## V

validateObject() (nPYc.objects.Dataset method), 22  
 validateObject() (nPYc.objects.MSDataset method), 30  
 validateObject() (nPYc.objects.TargetedDataset method), 38  
 ValuesOnly (nPYc.enumerations.DatasetLevel attribute), 88  
 VariableType (class in nPYc.enumerations), 87