# **Code Documentation – Dynamic Binning**

***Pre-Processing***

**Normalizing the Data:** Common for NMR, FTIR, and GPC Data

A screenshot of a computer code

Description automatically generated

This function:

1. Reads the data from the specified file path.
2. Separates the intensity and PPM (parts per million) data based on column names.
3. Normalizes the intensity data so that each spectrum ranges from 0 to 1.

**‘nmr\_intensity.min (axis=1)’** – This function computes the minimum value of ‘nmr intensity’, axis=1 parameter specifies that the operation to be performed across columns for each row (i.e., for each sample or spectrum).

‘.**subtract(nmr\_intensity.min(axis=1), axis=0)’** – This subtracts the minimum intensity value from all corresponding values in each row of ‘nmr\_intensity’. The subtraction is broadcasted across each column of the row, effectively shifting the lowest value of each spectrum to 0. Axis=0 parameter ensures subtraction aligns correctly with each row’s minimum value.

**‘nmr\_intensity.max(axis=1)–nmr\_intensity.min(axis=1)’** - This expression calculates the range of the intensity values for each spectrum. It does this by subtracting the minimum intensity value of each row from the maximum intensity value of the same row. The result is a Series where each entry is the range of intensity values for each spectrum.

**‘.div(nmr\_intensity.max(axis=1)–nmr\_intensity.min(axis=1), axis=0)’** – This division operation scales each value in the spectrum such that the lowest value becomes 0 and the highest value becomes 1. The division is broadcasted across each row, normalizing the data within the range [0,1]. This step is crucial for ensuring that each spectrum's intensity values are on a comparable scale, particularly important when analyzing multiple spectra together.

**Spectral Processing Loop:**A screenshot of a computer code

Description automatically generated

The main loop iterates through each normalized spectrum to process the data.

**‘spectrum = normalized\_intensity.iloc[sample\_index, :].copy(); spectrum.copy(); ppm\_values = nmr\_ppm.iloc[sample\_index, :]’** – Stores the y-axis and x-axis values in separate variables.

**find\_peaks** function from **SciPy. Signal** library is used to detect peaks in the spectroscopy data.

‘find\_peaks’ is designed to identify the local maxima in a dataset by comparing each point to its neighbors.

Parameters used in the code:

**‘spectrum’** – Input data for the function, in this case the intensity values.

**‘prominence’** - The prominence of a peak is a measure of how much it stands out due to its intrinsic height and its location relative to other peaks. A high prominence means the peak is much higher than the points around it.

Prominence, in mathematical terms calculates the vertical distance between the peak and its lowest contour line. The contour line is defined as the higher of the lowest points on either side of the peak that are lower than the peak. This parameter is crucial in distinguishing true peaks from the noise.

**‘find\_peaks’ function:**

1. Function initially identifies all local maxima, a point that is higher than the points immediately before and after it.
2. For each local maximum, the algorithm assesses its prominence based on the specified prominence value. It calculates how much higher the peak is compared to the highest of the two points at which the signal drops on either side of the peak by at least the prominence value. Peaks that do not meet this prominence criterion are discarded.
3. Output: The function returns indices of the array where peaks were identified. These indices correspond to the position of the peaks in the data array, spectrum.

The code iterates over each spectrum in the dataset. For each spectrum, it identifies peaks using find\_peaks with a very small prominence value, suggesting that even very slight elevations in intensity compared to adjacent values are considered as peaks to capture the subtle differences for the spectroscopy data where those differences can mean different compounds, or functional groups.

After identifying the peaks, the above code also calculates the widths of these peaks using **‘peak\_widths’** function, with a rel\_height of 0.995, which measures how wide each peak is at 99.5% of its height.

**‘widths = peak\_widths(spectrum, peaks, rel\_height=0.995)’ –**

Parameters used in the code:

**‘spectrum’** – Input data for the function, in this case the normalized intensity values.

**‘peaks’** – Indices of peaks, stored in the variable **‘peaks’** from find\_peaks function.

**‘rel\_height’** - relative height at which the peak width is measured as a percentage of its prominence.

Output:

‘peak\_widths’ function returns 4 arrays, and it is stored in the variable **‘widths’**.

**widths[0]** – The widths for each peak in samples. These widths are typically measured as distance between the points on the left and right where the signal ascends and descends to the height defined by ‘rel\_height’.

**widths[1]** – The height of the contour lines at which widths were evaluated.

**widths[2]** - Interpolated positions of left intersection point of a horizontal line at the respective evaluation height.

**widths[3]** - Interpolated positions of right intersection point of a horizontal line at the respective evaluation height.

**Identifying & removing the water peak:**

A screenshot of a computer program

Description automatically generated

Removing the water peak is specific for NMR. This approach effectively deals with the water peak, which can obscure or distort the interpretation of other chemical shifts in NMR spectroscopy. By minimizing this peak, the spectral data becomes cleaner and more focused on the chemical components of interest, facilitating more accurate analyses.

A close up of words

Description automatically generated

In this code, we assume the most prominent peak (based on intensity, not on prominence) is likely the water peak because the water peak generally stands out in aqueous solutions.

Identifying index of the maximum peak aka water peak -

**‘max\_peak\_index = peaks[np.argmax(spectrum.iloc[peaks])]’ –**

**peaks:** This array contains indices of the peaks found in the spectrum using the find\_peaks function.

**spectrum.iloc[peaks]:** This extracts the intensity values at the indices specified in peaks, effectively giving you the intensity values at each peak.

**np.argmax(spectrum.iloc[peaks]):** This function returns the index of the highest value in the spectrum.iloc[peaks] array. Note that the value returned is relative to the start of the peaks array, not the original spectrum array.

**peaks[...]:** By placing ***np.argmax(spectrum.iloc[peaks])*** inside **peaks[...]**, you translate the relative index back to an index of the original spectrum array. This gives you the index of the highest peak within the entire spectrum.

Identifying width of the water peak -

**‘max\_peak\_width = just\_widths[np.argmax(spectrum.iloc[peaks])]’ -**

**just\_widths:** This array contains the widths of each detected peak at a specified relative height (from peak\_widths function).

**np.argmax(spectrum.iloc[peaks]):** As explained above, this finds the index of the maximum peak.

**just\_widths[...]:** Retrieves the width of the most prominent peak by using the index found with np.argmax.

A screenshot of a computer code

Description automatically generated

**math.ceil(max\_peak\_index - max\_peak\_width):** Calculates the left boundary of the water peak. math.ceil is used to ensure the boundary is a whole number and doesn't start in between data points. The calculation determines how far left from the peak's maximum point you go based on the peak's width.

**math.floor(max\_peak\_index + max\_peak\_width):** Similarly, calculates the right boundary of the water peak. math.floor ensures the boundary is also a whole number, rounding down so you don't end beyond an actual data point. This determines how far right from the peak's maximum point you extend.

**spectrum[left\_range:right\_range] = min(spectrum):** Sets the intensity values of the spectrum from left\_range to right\_range (the region around the peak) to the minimum intensity found in the entire spectrum. This effectively "flattens" or minimizes the peak, reducing its prominence.

By setting the intensities in the range of the water peak to the spectrum’s minimum value, you effectively reduce interference caused by this dominant peak, allowing for a clearer analysis of other chemical shifts and peaks in the spectrum. This step is critical in preprocessing NMR data, especially when the water peak can obscure or distort other spectral features.

**Identifying peaks in the dataset for dynamic binning:**

A screen shot of a computer code

Description automatically generated

This above code essentially performs all the tasks that was described earlier but, on the spectrum, where the water peak’s removed.

Since we normalized the intensity values once again after removing the water peak, we chose a lower prominence value of 0.001 to identify only the significant peaks.

***Post Processing***

A screen shot of a computer

Description automatically generated

This function sums up the intensities provided in the data frame, whole\_spectra\_df, across 12 samples.

**‘axis=0’** parameter sums the intensities along the columns, effectively combining all the individual spectra into a single spectrum that represents the sum of intensities at each chemical shift (ppm) across all samples.

A screenshot of a computer code

Description automatically generated

Function, bin\_around\_peaks:

This function dynamically creates bins around peaks identified in NMR spectra (similarly for FTIR and GPC). The bins are defined based on the peak widths, and the area under each peak within these bins is calculated using the trapezoidal rule.

Parameters:

**whole\_spectra\_df:** Data Frame containing the spectra data from which intensities are taken. Here we are using the data frame that has been pre-processed after removing the water peak and normalizing the data.

**ppm\_values\_df:** Data Frame containing the corresponding ppm (chemical shift) values for each spectrum.

**widths:** Array-like structure containing the widths of the peaks as determined by peak\_widths.

A list, **binned\_data\_list**, is initialized to store the binned data for each spectrum.

A computer code with black and blue text

Description automatically generated

Loop Over Each Spectrum:

For each spectrum in whole\_spectra\_df, it extracts intensity and ppm values.

Initializes a list, **binned\_sample\_data**, to hold the binned intensities for the current spectrum.

**A screenshot of a computer code

Description automatically generated**

**Peak-Based Binning:**

For each peak in no\_water\_peaks:

Determines the bin width **(window\_width)** as the width of the peak (from widths).

Calculates the start **(window\_start)** and end **(window\_end)** indices of the bin around each peak.

Computes the area under the curve within this bin using **np.trapz**, which integrates using the trapezoidal rule.

Appends the computed area to **binned\_sample\_data**.

**Store Binned Data:** Appends the binned\_sample\_data for each spectrum to binned\_data\_list.

**Return Data Frame:** Converts binned\_data\_list into a Data Frame, where each column represents a binned peak area, labeled as Bin\_i for each peak i.

***Regression Analysis***

A screenshot of a computer code

Description automatically generated

The above lines of code pertain to the application of Principal Component Analysis (PCA) on the binned NMR data, similarly for GPC and FTIR Data.

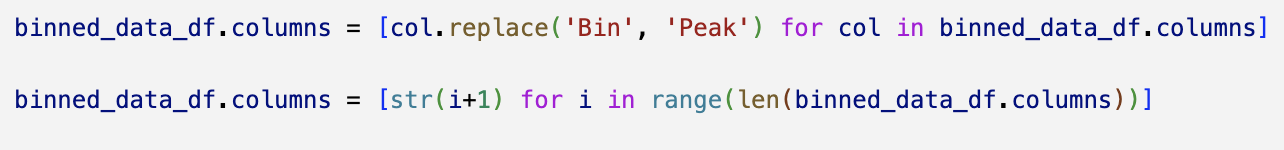
**binned\_data\_df.columns = [col.replace('Bin', 'Peak') for col in binned\_data\_df.columns]:** This line changes the column names from 'Bin' to 'Peak' to better reflect what the columns represent, indicating that these are specific spectral peaks rather than just numerical bins.

**binned\_data\_df.columns = [str(i+1) for i in range(len(binned\_data\_df.columns))]:** This line simplifies the column names further by numbering them sequentially. This renaming could be intended to ease referencing in analysis outputs and graphs.

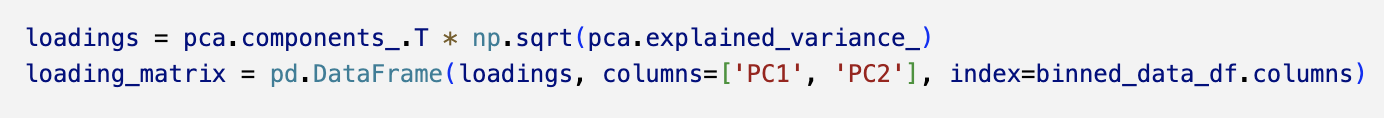
**PCA(n\_components=2):** Initializes a PCA model to reduce the dimensionality of the data to two principal components. This helps in capturing the most significant variance in the data with fewer dimensions, facilitating easier visualization and analysis.

**fit\_transform(binned\_data\_df):** This method fits the PCA model to the binned data and then transforms it into the principal components. The transformation re-expresses the data in terms of the directions (principal components) that capture the most variance.

**principal\_df:** The resulting principal components for each sample are stored in a Data Frame with columns labeled 'PC1' and 'PC2', representing the first and second principal components, respectively.

****

The first line replaces the word 'Bin' with 'Peak' in each column name to reflect each column represents peak rather than the bin, this is just for visual presentation.



**pca.components\_.T:** This retrieves the principal axes in feature space, showing the directions of maximum variance (the eigenvectors). Transposing these components prepares them for multiplication with the square root of the explained variance.

**np.sqrt(pca.explained\_variance\_):** Computes the square root of the variance explained by each principal component. This step is part of calculating the loadings, which measure how much each original variable contributes to each principal component.

**loading\_matrix:** Constructs a Data Frame containing the loadings, which are the coefficients of the original variables (in this case, the peak areas) on the principal components. This matrix is helpful to interpret which peaks (variables) have the most influence on the components, providing insights into the underlying structure of the data.

A screenshot of a computer program

Description automatically generated

Load and Prepare Performance Data:

**Load Data:** pd.read\_csv() reads the CSV file containing performance metrics related to NMR data, setting the 'Sample' column as the Data Frame index.

**Extract Target Variable:** The 'Performance (mean)' column, which likely represents some quantitative measure of performance (e.g., chemical yield, reaction rate, etc.), is extracted as the dependent variable for the regression model.

**Reset Index:** reset\_index(drop=True) ensures that the series Y has a default integer index. This is useful for alignment when performing operations with other data arrays or series without index conflicts.

**Polynomial Transformation:** Although the code sets degree=1, essentially keeping it linear, using Polynomial Features can be easily adjusted for higher-degree polynomial regression by changing the degree parameter.

**Fit and Transform:** This step transforms the PCA output (presumably stored in principal\_df with principal components as features) into polynomial features. For degree=1, this adds an intercept term (a column of ones) to the feature set.

A screenshot of a computer

Description automatically generated

**R2 Score:** Measures the proportion of variance in the dependent variable that is predictable from the independent variables.

**Mean Absolute Error (MAE):** The average magnitude of the errors in a set of predictions, without considering their direction.

**Mean Squared Error (MSE):** The average of the squares of the errors—that is, the average squared difference between the estimated values and the actual value.

A close-up of a computer code

Description automatically generated

**Add Constant:** While Polynomial Features with degree=1 already includes an intercept, using sm.add\_constant() ensures compatibility with stats models' OLS which does not include an intercept by default.

**Ordinary Least Squares (OLS) Model:** Fits an OLS regression model using stats models, a module providing classes and functions for the estimation of many different statistical models.

**Model Summary:** The .summary() provides a full overview of the regression results, including things like the coefficient values, R-squared, adjusted R-squared, F-statistic, Log-likelihood, AIC/BIC, and much more, offering deep insights into the model's performance and statistical significance.