# **Code Documentation – Naive Binning**

***Pre-Processing***

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

**A screen shot 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 & Identifying the water peak:**

This step is specific for NMR data and in this step using **‘find\_peaks’** function, water peak is removed from the spectra by minimizing the water peak. This approach effectively deals with water peak, which can obscure or distort the interpretation of other chemical shifts in NMR spectroscopy.

**A screenshot of a computer program

Description automatically generated**

The main loop iterates through normalized y-axis values for each sample 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 water peak in the NMR 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.

**A screen shot of a computer code

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 close-up 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.

**Performing ANOVA function:**

Analysis of Variance (ANOVA) is statistical technique used to analyze the differences among group means in a sample. In the context of spectroscopy data, ANOVA can be used to determine whether there are statistically significant differences in spectral responses across different materials. This helps in determining the responses that essentially do not carry any special feature.



**‘perform\_anova’** function is designed to process the spectroscopy data for ANOV analysis by organizing it into bins and preparing a structured Data Frame for statistical testing. The function takes three parameters: X-axis(nmr\_ppm), Y-axis(nmr\_intensity), and bin\_size.

**Initializing Variables:**

**total\_points:** Retrieves the total number of data points in the dataset, stored across the columns.

**num\_bins:** Calculates how many complete bins of size bin\_size can be formed from the total data points.

**remaining\_points:** Determines if there are any leftover points after forming complete bins that do not fill a full bin size.

**anova\_results:** Initializes an empty Data Frame intended to store the results of the ANOVA analysis (although in this snippet, it is not used beyond initialization).

**anova\_Data:** Initializes an empty list to store data for ANOVA analysis (used later for Data Frame construction).

**Function enters a loop to process each bin:**

A computer screen shot of a code

Description automatically generated

**Bin Ranges:**

* **start:** Calculates the starting index for the current bin.
* **end:** Calculates the ending index for the current bin. If processing the last bin and there are remaining points, it adjusts to include all remaining points.

**Data Extraction:**

A screen shot of a computer code

Description automatically generated

**current\_bin\_intensity:** Extracts intensity values from the nmr\_intensity Data Frame for the current bin range and flattens the array for easier handling.

**current\_bin\_size:** Computes the size of the current bin, which is crucial for correctly repeating indices and ppm values.

**current\_bin\_ppm:** Replicates the PPM values associated with each point in the current bin. It uses np.tile to repeat the PPM values as necessary to match the number of intensity measurements. This array is then sliced to match the length of **current\_bin\_intensity**.

**current\_bin\_materials:** Repeats the sample/material indices for each data point in the current bin to correspond to the flattened intensity values. This repetition is important for statistical analysis, ensuring each data point has an associated material label.

**Modified Data frame for ANOVA analysis:**

Constructs a Data Frame anova\_data for the current bin that includes:

**'Intensity':** Flattened array of intensity values for the current bin.

**'PPM':** Corresponding PPM values for each intensity measurement.

**'Material':** Sample or material identifiers for each data point.

A screen shot of a computer code

Description automatically generated

After preparing the anova data frame for each bin, which includes, X-axis(PPM), Y-axis(Intensity), and ‘Material’, the function then performs ANOVA analysis for each bin:

**ANOVA Setup:**

**‘anova = AnovaRM(...)’:** This line initializes a Repeated Measures ANOVA analysis using the statsmodels library’s AnovaRM class. The parameters are:

**‘data**’=anova\_data: The Data Frame containing the data for the current bin.

**‘depvar='Intensity'**: Specifies that the dependent variable (the measurement of interest) is 'Y-axis(Intensity)'.

**‘subject='Material'**: Defines 'Material' as the subject variable, which means that data are grouped by 'Material' and each subject might be exposed to different conditions within the same test.

**‘within=['PPM']’**: Sets ‘X-axis(PPM)' as the within-subject factor, indicating that each subject (material) has multiple measures taken at different PPM values.

**‘.fit()’**: This method fits the ANOVA model to the provided data, computing the necessary statistical measures to determine the significance of the observed differences.

Results Data frame:

After fitting the ANOVA model, the function extracts key results from anova.anova\_table and appends them to the anova\_results Data Frame. This includes:

**'Bin'**: The current bin number, incremented by 1 for human readability.

**'F-Value'**: The F-statistic value from the ANOVA, indicating the ratio of variance between groups to variance within groups.

**'P-Value'**: The p-value associated with the F-statistic, used to determine if the observed differences in means across different PPM levels are statistically significant.

**'DF Within' and 'DF Between'**: The degrees of freedom within the groups and between the groups.

The function loops through all specified bins, performs ANOVA for each, and collects the results into anova\_results, which provides a comprehensive view of the statistical analysis across all bins.

The final output **(‘anova\_results’)** can be used to identify which bins show significant differences in ‘Y-axis' across varying 'X-axis' levels for different materials, providing valuable insights into the spectral characteristics of the materials studied.

**Note:**

We were unable to perform ANOVA on GPC dataset. We kept running into this error of Data being unbalanced.

Unsure if it’s due to the precision range in the dataset that was given to us that is raising the error when we are using ANOVARM function. So, we proceeded with using PCA on GPC dataset directly after normalizing data without doing ANOVA.

A table with numbers and a few minutes

Description automatically generated

**Processing significant bins from ANOVA Analysis:**

A screenshot of a computer code

Description automatically generated

**‘extract\_data\_from\_signficant\_bins’ -** To extract data from bins that have been identified as statistically significant based on the results of an ANOVA analysis. This is typically used to focus subsequent analyses on parts of the data that show significant differences, thereby reducing noise and improving the reliability of the findings.

**Input Parameters:**

**data:** The full dataset, assumed to be a Data Frame where each row represents a sample and each column a different measurement across a continuous scale (like time, wavelength, etc.).

**significant\_bins:** A Data Frame containing information about bins that have been determined to be statistically significant from an ANOVA. This includes which bins (by index) show significant differences.

**bin\_size:** The size of each bin in terms of the number of columns in the data Frame.

**extracted\_data:** Initializes an empty list that will store the extracted data for each sample.

The outer loop iterates over each sample in the data Data Frame.

**sample\_data:** A temporary list that collects data for the current sample across all significant bins.

A screen shot of a computer code

Description automatically generated

The nested loop iterates over each row in significant\_bins, which contains the indices of bins deemed significant by ANOVA.

**start\_index and end\_index:** Calculate the start and end indices for the current bin in the data Frame based on the bin's index and the specified bin\_size.

**current\_bin\_data:** Extracts the segment of the data corresponding to the current significant bin for the current sample.

**sample\_data.extend(current\_bin\_data.values):** Extends the sample\_data list with the values from the current significant bin. This flattens the data into a single list for each sample, combining data from multiple significant bins.

After processing all bins for a sample, the compiled sample\_data is appended to extracted\_data.

This results in **extracted\_data** containing one list per sample, where each list contains the combined data from all significant bins for that sample.

Converts **extracted\_data** (a list of lists) into a pandas Data Frame. Each column in the resulting Data Frame represents a measurement point from the significant bins, named sequentially as Intensity\_0, Intensity\_1, ..., etc.

**Returns:** A Data Frame where each row corresponds to a sample, and each column corresponds to data extracted from the significant bins. The columns are named to reflect that they contain intensity data from various points, helping in the identification of which part of the original dataset each column comes from.

**significant\_bins = anova\_results[anova\_results['P-Value'] < 0.05]:** Filters the anova\_results Data Frame to include only those bins where the p-value is less than 0.05, indicating statistical significance.

**Binning the Data from Significant bins:**

A screenshot of a computer program

Description automatically generated

**‘extract\_and\_bin\_data’** function calculates the area under the curve for the segments that has been determined to be significant based on ANOVA analysis.

**binned\_areas:** A list to store the areas under the curve for each sample across all significant bins.

The function loops through each sample in the dataset (data). Each sample might represent a different experiment or measurement session.

**sample\_areas:** A list that collects the area under the curve for each significant bin within the current sample.

For each significant bin (identified previously and provided in **significant\_bins**), the function:

**Bin Indices Calculation:**

**start\_index:** Calculates the starting column index for the current bin in the dataset.

**end\_index:** Determines the ending column index, ensuring it does not exceed the number of columns available in data.

**Data Extraction:**

**current\_bin\_data:** Extracts the data for the current bin from the sample.

**current\_bin\_ppm:** Extracts the corresponding PPM (parts per million) values, which serve as the x-axis values necessary for calculating the area under the curve.

Uses the np.trapz function from the NumPy library to compute the area under the curve. This function integrates using the trapezoidal rule, which is a numerical integration method ideal for data that forms a curve or when data points are obtained experimentally.

**Parameters for np.trapz**:

**current\_bin\_data:** y-values for the integration.

**current\_bin\_ppm.iloc[0]:** x-values for the integration, assumed to be consistent across all rows for each bin.

Each area calculated is appended to sample\_areas, which after iterating through all significant bins, gets appended to binned\_areas, thus collecting all area under the curve values for each sample.

Converts **binned\_areas** (a list of lists) into a pandas Data Frame, which provides a structured format with columns labeled as **Area\_Bin\_1**, **Area\_Bin\_2**, etc., reflecting each significant bin’s area.

The number of columns (num\_columns) in the resulting Data Frame matches the number of significant bins processed for each sample.

The function is called with **Y-axis(whole\_spectra\_df)**, **X-axis(nmr\_ppm)**, significant\_bins, and bin\_size, which are:

**Y-Axis(whole\_spectra\_df):** The complete dataset containing the spectral data.

**X-Axis(nmr\_ppm):** The dataset containing the X-axis(PPM) values corresponding to the data points in whole\_spectra\_df.

**significant\_bins:** A Data Frame or similar structure that lists bins identified as significant.

**bin\_size:** The number of data points in each bin.

**binned\_data:** A Data Frame where each row corresponds to a sample and each column to the area under the curve of a significant bin, effectively condensing the spectral data into quantifiable metrics based on previously identified significant regions of the spectrum.

***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 GPC data, with a similar approach for NMR and FTIR Data.

**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):** 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.

**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 screen shot 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.

**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.

**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.