2nd PANHELLENIC CONGRESS OF MEDICAL PHYSICS
4-6 OCTOBER 2024| EUGENIDES FOUNDATION

A versatile and intuitive workflow for the analysis of Raman spectra using Python

Eleftherios Pavlou[1], Nikolaos Kourkoumelis[1]

[1]Laboratory of Medical Physics, School of Medicine, University of Ioannina

Ευρωπαϊκή Ένωση
European Social Fund

Operational Programme
Human Resources Development,
Education and Lifelong Learning

Co-financed by Greece and the European Union

ΕΣΠΑ 2014-2020
ανάπτυξη - εργασία - αλληλεγγύη

1. Background-Aim

Raman spectroscopy

- Raman spectroscopy is a vibrational spectroscopic technique that can provide information about the molecular composition and structure of a material by studying how inelastically scattered (Raman scattered) light interacts with the vibrational modes of the material's molecules.
- Due to its advantages, it has seen wide use in biomedical applications, from detecting and monitoring diseases to studying cells and analyzing drugs.

Advantages

- Non-destructive
- Minimal to no sample preparation
- Rapid
- Minimal interference from water (suitable for wet tissues)
- Works with solids, liquids, and gases

Disadvantages

- Weak signal due to small scattering cross-section
- Strong background due to fluorescence
- Complex spectra with overlapping bands
- Preprocessing required before analysis
- Raman experiments can generate large amounts of data

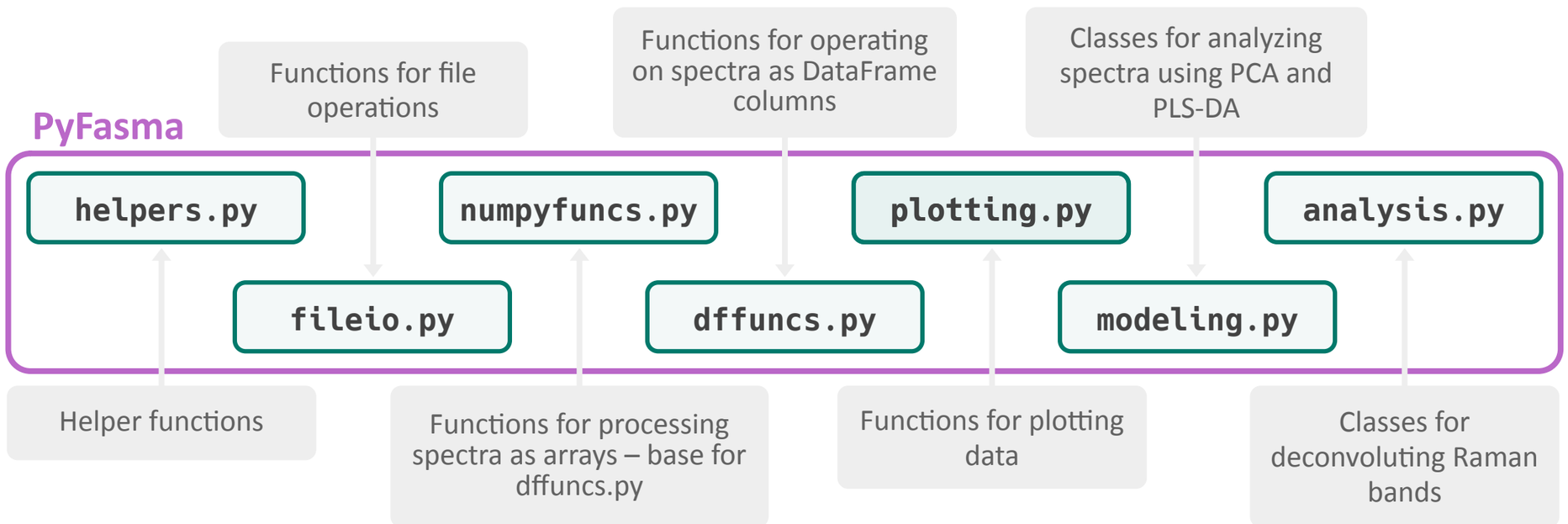
Our work

We present an efficient, versatile, and intuitive workflow for preprocessing and analyzing Raman data using the **free** and **open-source PyFasma** package. We developed the package with the goal of providing a high level programming interface that offers spectroscopists flexibility and ease of use, without requiring expert Python knowledge.

2. Materials & Methods

Package overview

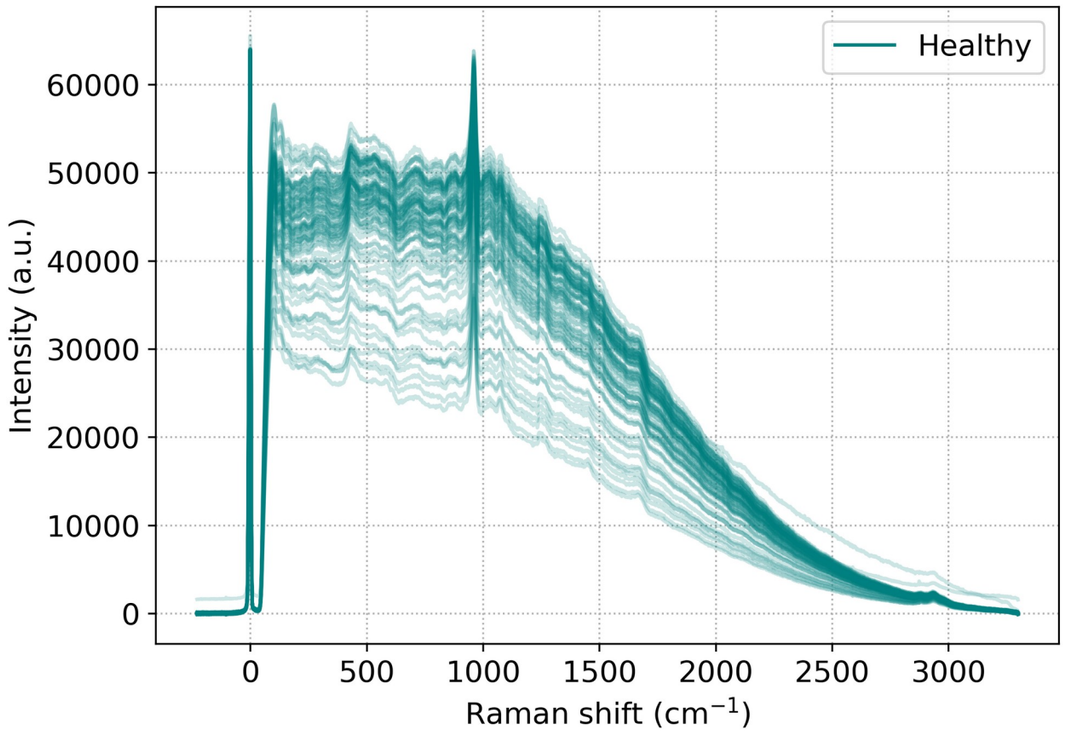
- PyFasma is based on **Python 3.12** and depends on several Python packages: **LMFIT**, **Matplotlib**, **NumPy**, **pandas**, **pybaselines**, **SciPy**, **scikit-learn**, **Seaborn**, **spc**.
- The package is created as a one-stop shop solution for conducting the whole analysis of Raman data using robust and trusted algorithms: from *file manipulation* and *preprocessing* of spectra to *multivariate statistical analysis* and band *deconvolution*.
- Central data structure is the **pandas DataFrame**, which PyFasma extends, and was chosen for its flexibility and data manipulation capabilities.
- PyFasma's structure is presented below:

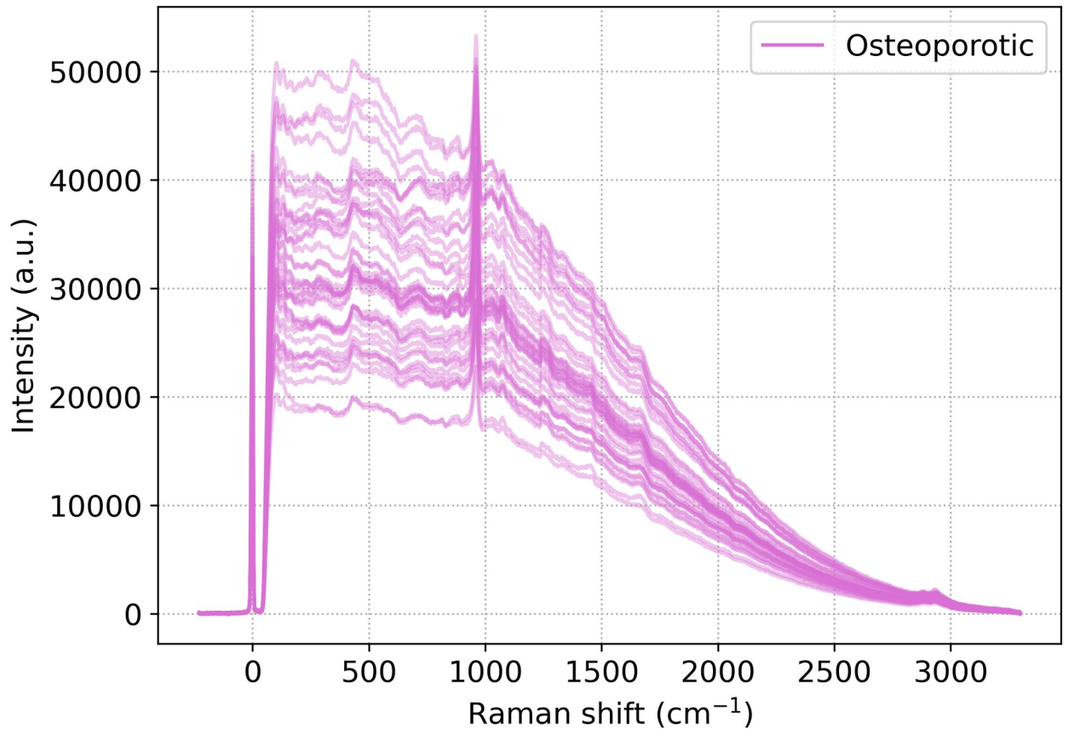


2. Materials & Methods

Samples for demonstration

- To demonstrate the workflow using PyFasma, we conducted a comparative analysis using 82 Raman spectra obtained from the tibias of healthy rabbits and 40 spectra from rabbits with induced osteoporosis (122 Raman spectra in total).
- Spectra were collected using a BWTEK i-Raman Plus spectrometer, operating at 785 nm, with a power output of 200 mW at the probe and signal collection time of 6 s.
- The unprocessed spectra for the two groups are presented below.





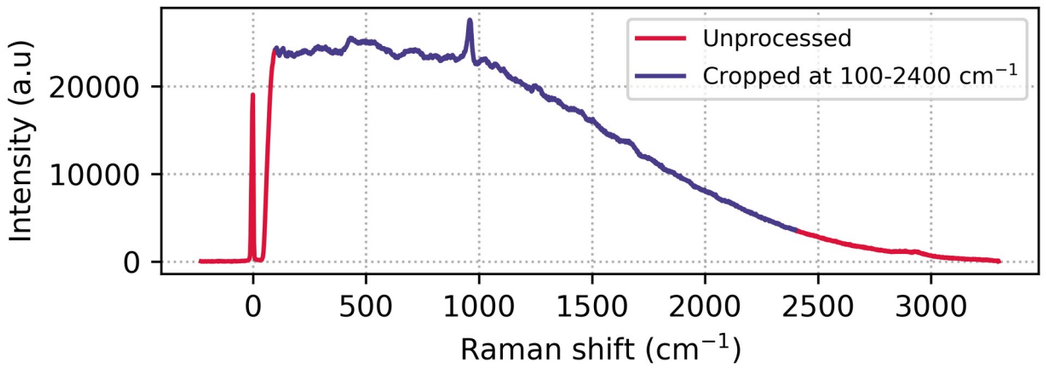
3. Results

Preprocessing

- After importing the `dffuncs` module (`from pyfasma import dffuncs`), we preprocess the spectra (columns) in a DataFrame df as follows: (1) **initially crop** the spectra to a wider region of interest (100-2400 cm^{-1}), (2) **remove spikes** and **smooth** using a Savitzky-Golay filter, (3) **baseline correct** using the SNIP algorithm, (4) **normalize** to the phosphate peak intensity (max between 950-970 cm^{-1}), and **finally crop** to the fingerprint region (400-1800 cm^{-1}).

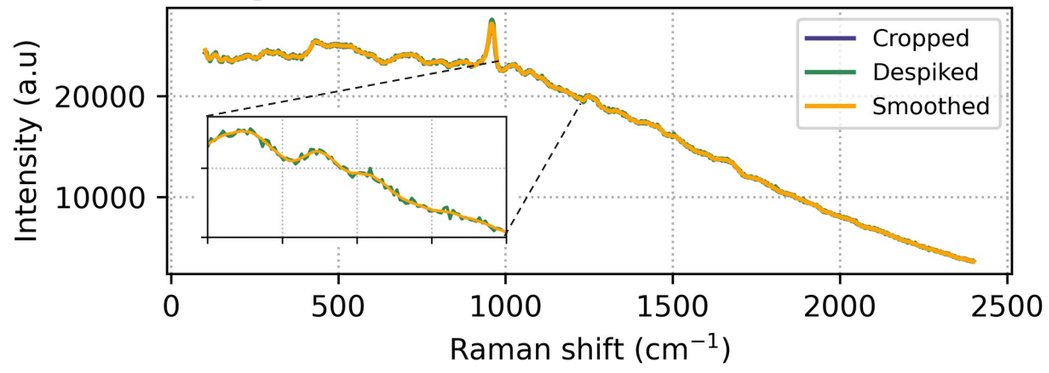
(1)

```
df_cropped = df.pyfasma.crop(xrange=[100, 2400])
```



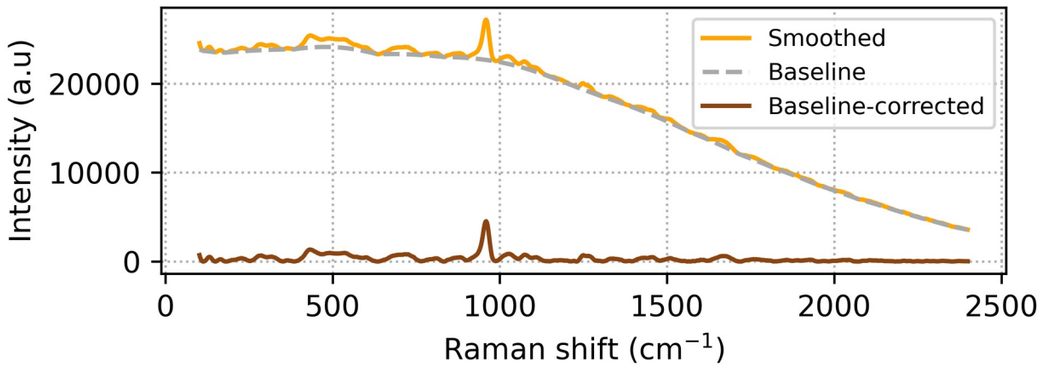
(2)

```
df_despiked = df_cropped.pyfasma.despike(threshold=1000)
df_smoothed = df_despiked.pyfasma.smooth(params=[15, 3],
    kind='savgol')
```



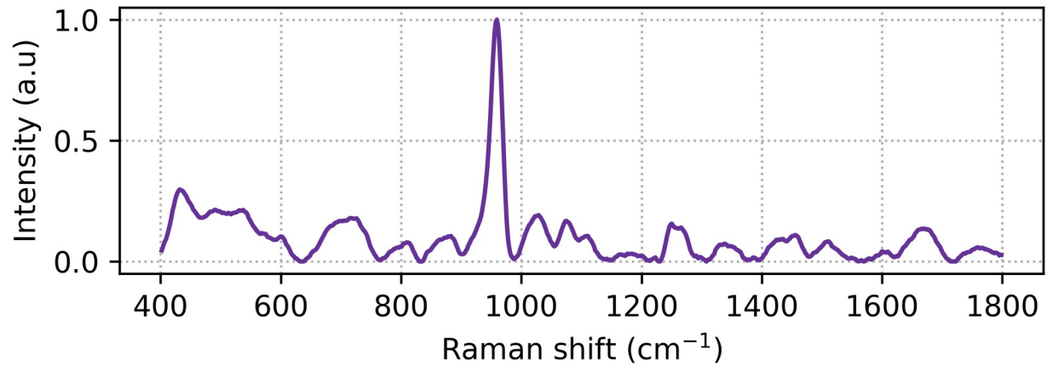
(3)

```
df_baselined = df_smoothed.pyfasma.baseline_correct(
    kind='snip', decreasing=True, max_half_window=20)
```



(4)

```
df_normalized = df_baselined.pyfasma.normalize(
    kind='intensity', xrange=[950, 970])
df_final = df_normalized.pyfasma.crop(xrange=[400, 1800])
```



3. Results

Principal Components Analysis (PCA)

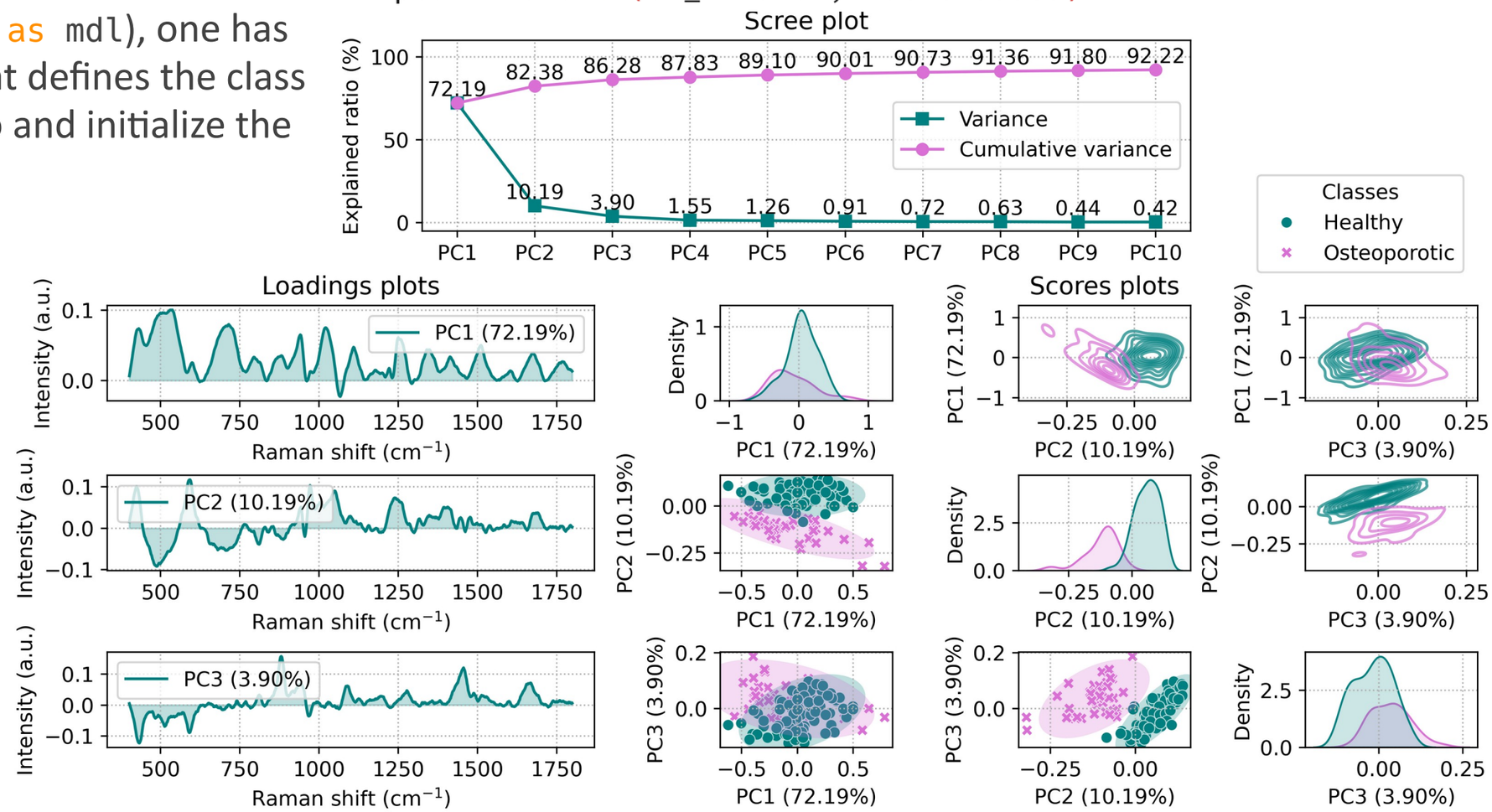
- PCA is an **unsupervised multivariate statistical analysis** technique that is widely used in Raman analysis both for **dimensionality reduction** and for uncovering helpful **insights** about the **relationship** between variables.

- To perform PCA with PyFasma, after importing the `modeling` module (`import pyfasma.modeling as mdl`), one has to just assign the `hue` list that defines the class that each sample belongs to and initialize the `PCA` class.

```
classes = ['Healthy' if 'Rbhf' in col else 'Osteoporotic'
    for col in df_final.columns]
pca = mdl.PCA(df_final.T, hue=classes)
```

- The scree plot shows that the first three PCs explain most (86%) of the observed variance.

- By examining the scores plots, good separation is observed along PC2 and PC3. The respective loadings plots indicate the variables that contribute more to the variance.

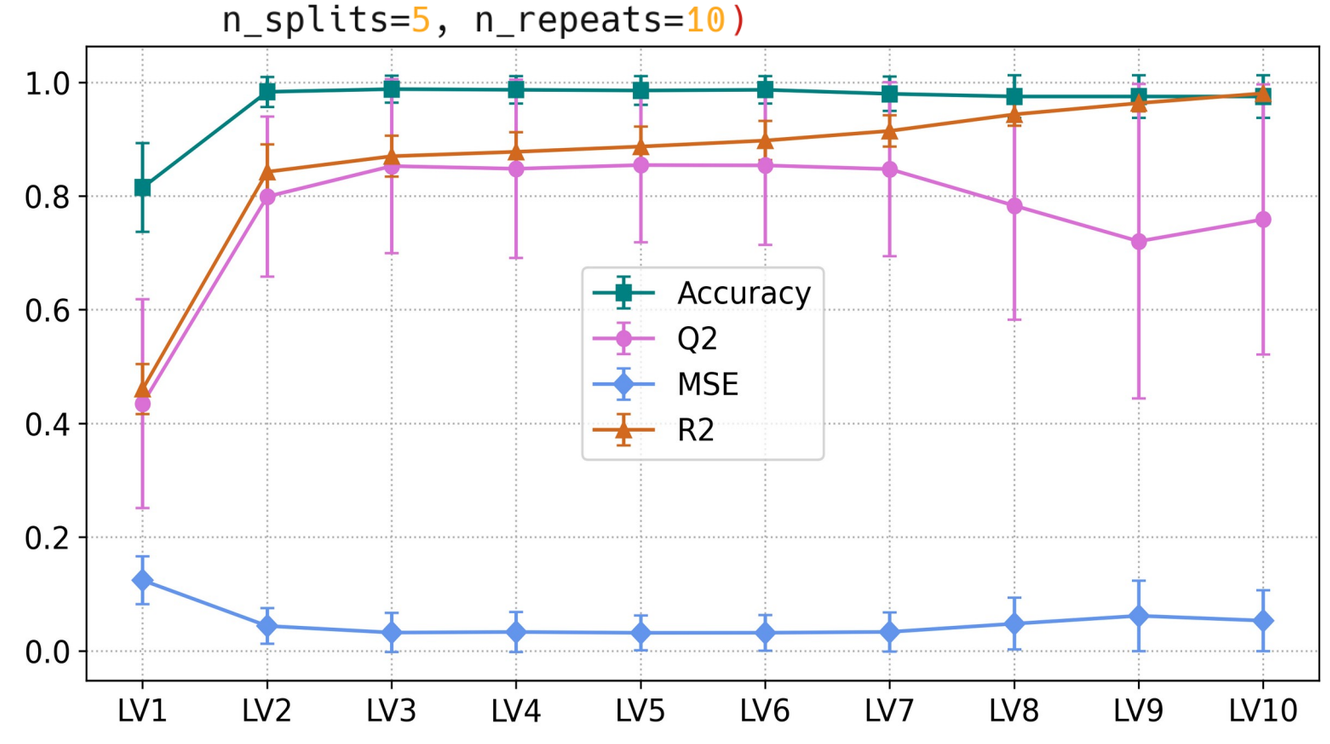


3. Results

Partial Least Squares Discriminant analysis (PLS-DA) (1)

- PLS-DA is a **supervised multivariate statistical analysis** technique that is used both for **dimensionality reduction** and for **classification**. It has emerged as an adaptation of PLS regression that can handle categorical response variables.
- As a supervised technique, the data must be split to train and test datasets prior to creating a model. We used the same `classes` list we assigned in PCA as the response variables and split the data to a 70/30 ratio in a stratified fashion using scikit-learn's `model_selection.train_test_split` method.
- Before creating a PLS-DA model, the optimal number of components must be determined so that the model works without been over- or under-fitted. This can be achieved with performing **cross-validation** by setting `cross_val=True` when initializing the PLS class of the `modeling` module. We opted for repeated (`n_repeats=10`) stratified (`stratify=True`) cross-validation with 5 KFolds (`n_splits=5`).
- Based on the resulting cross-validation metrics plot, it can be determined that the optimal number of components for the PLS-DA model is 2.

```
pls = mdl.PLSDA(x_train, x_test, y_train, y_test,
    cross_val=True, n_components=10, stratify=True,
    n_splits=5, n_repeats=10)
```



3. Results

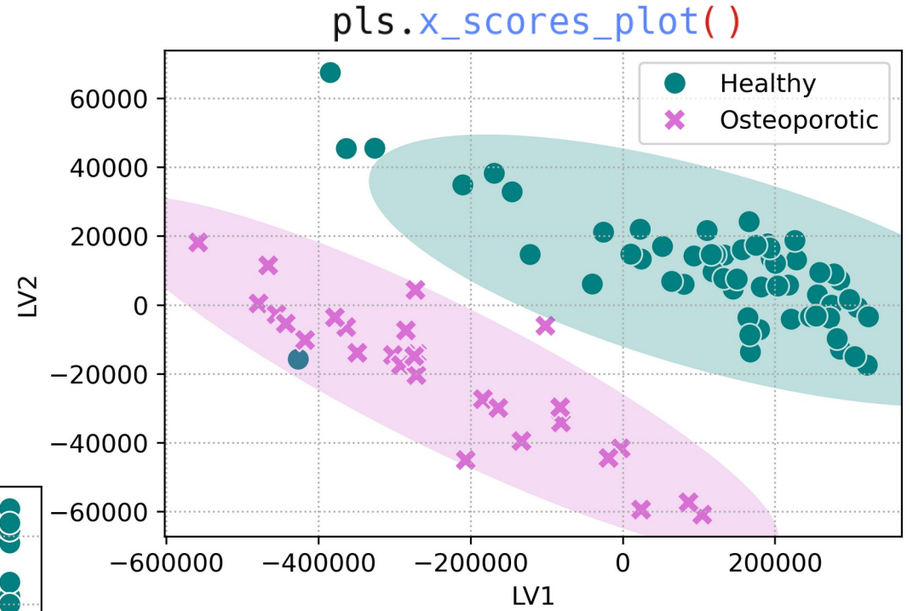
Partial Least Squares Discriminant analysis (PLS-DA) (2)

- After determining the number of components for the model, we can create it:

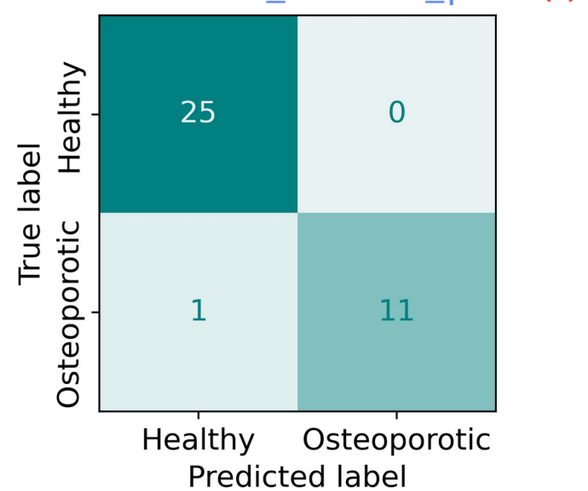
```
pls = mdl.PLSDA(x_train, x_test, y_train, y_test, n_components=2)
```

- The available visualizations present good discrimination between the two classes and the metrics indicate good prediction capabilities for the model.

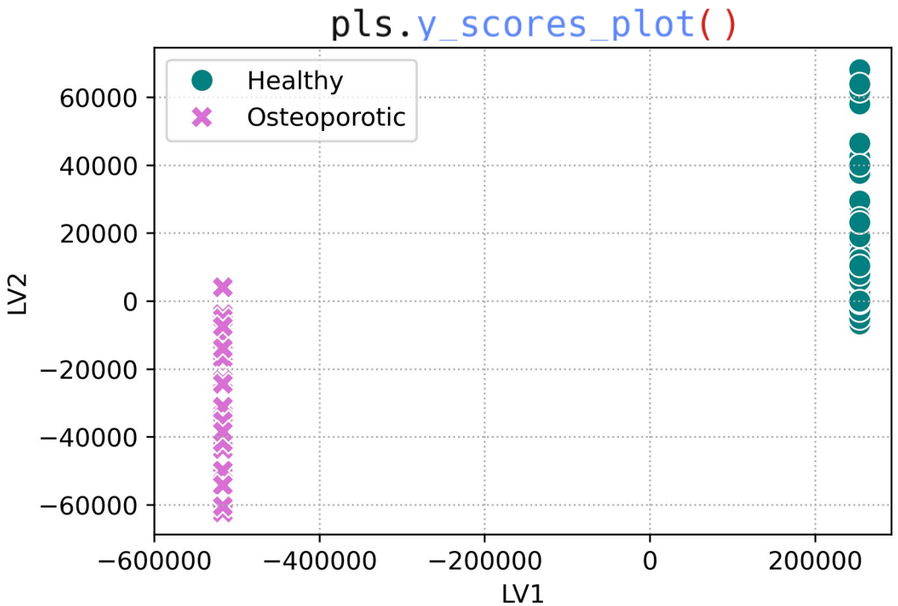
pls.x_scores_plot()



pls.confusion_matrix_plot()



pls.y_scores_plot()



pls.prediction_metrics_df

	Precision	Sensitivity	Specificity	F1 score	ROC-AUC	Accuracy
Healthy	0.961538	1.000000	0.916667	0.980392	0.973333	NaN
Osteoporotic	1.000000	0.916667	1.000000	0.956522	0.973333	NaN
Overall (macro)	0.980769	0.958333	0.958333	0.968457	0.973333	0.972973
Overall (micro)	0.972973	0.972973	0.972973	0.972973	0.983199	0.972973

4. Conclusions

- We presented the workflow of preprocessing and analyzing Raman spectra with PyFasma, a free and open-source Python 3 package.

- We showed the effectiveness of creating preprocessing pipelines with an intuitive and repeatable way.

- We also created, evaluated, and visualized unsupervised (PCA) and supervised (PLS-DA) multivariate models using only a few lines of code.

- Healthy and osteoporotic bones were well-separated in PCA, which is indicative of significant differences between the two classes.

- In PLS-DA, cross-validation was used for the determination of the optimal number of components for the predictive model. PLS-DA also provided good separation between the two bone classes and the evaluation metrics on the test data indicate good prediction capabilities for the model.