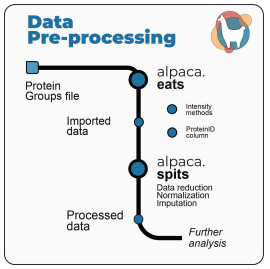
Feb 04, 2025

Pre-processing Proteomics Data with the Alpaca Pipeline



DOI
dx.doi.org/10.17504/protocols.io.dm6gp95j5vzp/v1

Borja Ferrero Bordera[1]

[1]LMU Munixh

Centrum Algatech

B Borja Ferrero Bordera
LMU Munixh

OPEN ACCESS

DOI: https://dx.doi.org/10.17504/protocols.io.dm6gp95j5vzp/v1

External link: https://github.com/borfebor/alpaca_proteomics/tree/main

Protocol Citation: Borja Ferrero Bordera 2025. Pre-processing Proteomics Data with the Alpaca Pipeline. **protocols.io** https://dx.doi.org/10.17504/protocols.io.dm6gp95j5vzp/v1

Protocol status: Working
We use this protocol and it's working

Created: January 13, 2025

Last Modified: February 04, 2025

Protocol Integer ID: 118213

Keywords: Proteomics, Mass-Spectrometry, Absolute Quantification, Bioinformatics, Python, Exoproteome, Extracellular proteins, alpaca, proteomics data with the alpaca pipeline data import, processing proteomics data, scale proteomics data, proteomics analysis, foundational steps in proteomics analysis, quantitative proteomic, absolute quantification in quantitative proteomic, proteomic, accuracy of protein quantification, alpaca pipeline data import, protein quantification, enabling meaningful biological insight, protein, standardized format for downstream analysis, meaningful biological insight, raw experimental data, dataset, reproducible result

Funders Acknowledgements:
People Programme (Marie Skłodowska-Curie Actions) of the European Union's Horizon 2020 Programme
Grant ID: 813979

Abstract

Data import and pre-processing are foundational steps in proteomics analysis, ensuring that raw experimental data is transformed into a clean, standardized format for downstream analysis. In the context of alpaca_proteomics, these steps are critical and applicable for both relative and absolute quantification in quantitative proteomics. Proper pre-processing removes contaminants, handles missing values, and normalizes datasets, minimizing technical variability and enhancing the accuracy of protein quantification. This ensures robust and reproducible results, enabling meaningful biological insights. The library streamlines these tasks, making it highly applicable for workflows involving large-scale proteomics data.

Troubleshooting

Getting Started

1 Install Alpaca library in case it has not been installed before

```
pip install alpaca-proteomics
```

2 Import the package

```
from alpaca_proteomics import alpaca
```

3 The following packages are recommended to be imported

```
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

Data import

4 Alpaca works with unprocessed proteomics datasets. The package takes the protein groups file, it has been tested with outputs from different search engines (MaxQuant, DiaNN, Spectronaut and MSFragger). As an example, we are working with the dataset from the exoproteome data published in **Ferrero-Bordera et al. 2024. Microbiology Spectrum**.

To import the data, the function ***alpaca.eats()*** returns a dataframe with the imported protein groups and has a first look at the conditions and intensity columns that the data contain.

```
file = 'proteinGroups.txt'

df, id_col, it = alpaca.eats(file)
```

The column Protein IDs was detected to contain your ProteinGroups IDs.
The following intensity methods were detected in the data: Intensity, iBAQ, LFQ

The function returned:

- **df** is the imported data as a pandas dataframe.
- **id_col** corresponds to the column which was detected to contain the Protein IDs
- **it** is a dictionary that groups the columns containing intensity data within each intensity method (e.g. LFQ) In our example, the data contained 3 intensity methods (Intensity, iBAQ, LFQ).

(Optional) Assistance on the quantification analysis

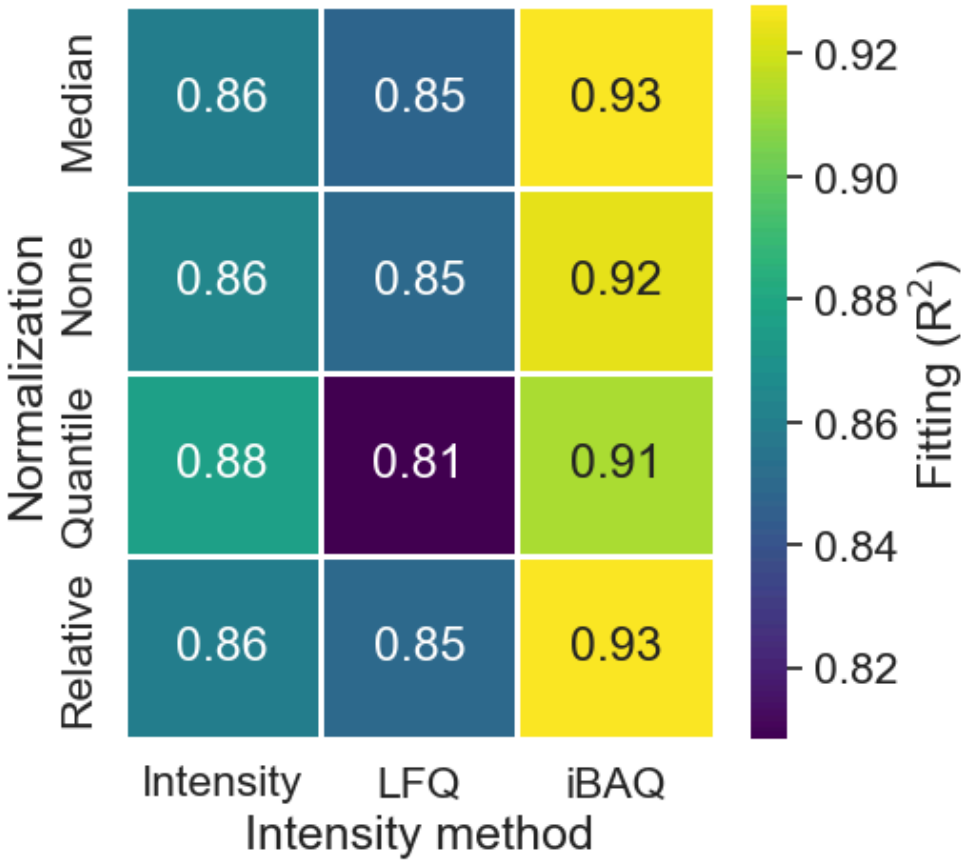
5 The framework includes a function to assess the most suitable intensity method for absolute quantification. For that, a file containing the anchor proteins used for quantification is needed.

The function ***alpaca.Consultant()*** accepts the imported protein groups dataframe, together with the anchor proteins dataframe and a dictionary listing all intensity methods and its respective columns (e.g. it). Additionally, the argument *added_samples* allows to restrict the inspection to a set of samples, in case not all samples were spiked with anchor proteins. Finally, the parameter *values_per_sample* serves to exclude those proteins that have not been identified in more than X replicates (this value should be between 0 and 1, this last one meaning that it should be in all replicates and 0 that there's no restriction).

```
# Path to the anchor proteins file
standards_file = 'UPS2.xlsx'

# Importation of the anchor proteins file (more details on these
are listed below)
st_proteins = alpaca.eats(standards_file)

# Samples in which anchor proteins were added
spiked_samples = ['Before_Induction_01', 'Control_01',
'Diamide_01']

# Valid values per condition
values_per_sample = 1/4

suggested = alpaca.Consultant(df,
                          st_proteins,
                          it,
                          added_samples=spiked_samples,
                          values_per_sample=values_per_sample)
```

5.1

```
# Visualization of the calculated fitting scores for each
intensity method
plt.figure(figsize=(5,5))
sns.set(font_scale=1.5)

sns.heatmap(suggested.pivot(index='Normalization',
                                          columns='Intensity
method',
                                          values='score'),
                                          annot=True,
                                          cmap='viridis',
                                          lw=1,
                                          cbar_kws={'label':
'Fitting (R$^2$)'})
```



Data pre-processing and formatting

6 Data pre-processing and formatting are crucial steps in proteomics data analysis, as they ensure the accuracy and reliability of downstream analyses. In the context of the Alpaca_proteomics Python library, these steps involve importing raw data, cleaning it by removing contaminants and decoys, and standardizing its structure to facilitate effective analysis. Proper pre-processing minimizes technical variability and enhances the quality of the results, leading to more robust biological interpretations.

This step is performed using the function ***alpaca.spits()*** as described below.

```
# Data pre-processing
values_per_sample = 1/4

clean_df = alpaca.spits(df,
                        lfq_method='iBAQ',
                        formatting=True,
                        valid_values=values_per_sample,
                        normalization='Median',
                        info_cols=['Accession', 'Gene names'])
```

Note

Parameters:
---
data : pandas.DataFrame
Input dataframe containing raw data for processing.
id_col : str
Column name in `df` representing unique identifiers for the dataset (e.g., 'Accession').
lfq_method : str
Column name or method representing LFQ (Label-Free Quantification) values in the data.
replicate_dict : dict
A dictionary mapping sample names to condition and replicate information.
cleaning : bool, optional, default=True
Whether to clean the data by removing contaminants and decoys.
formatting : str or bool, optional, default='auto'
Controls the format of the output dataframe ('auto', True, or False).
transformation : callable, optional, default=np.log2
A transformation function to apply to the data (e.g., log2).
normalization : str, optional, default='None'
Normalization method. Accepted values: 'None', 'Relative', 'Median', 'Quantile'.
valid_values : float, optional, default=0.7
Proportion of valid values required for each row to be retained.
imputation : str, optional, default=''
Method for data imputation. If empty, no imputation is performed.
****imp_kwargs** : dict
Additional keyword arguments passed to the imputation function.

This function reduces the complexity of the data to selected columns that contain the key information for quantitative proteomics experiments (Protein IDs, Sample IDs and intensity values). Additional columns could be added by passing them as a list in the ***info_cols*** argument of the function. More details on the function parameters are listed in the note above.

The resulting data will look somehow similar to the table below.

	Accession	Protein	Sample	iBAQ	Condition	Replicate
	C0SP82	YoaE	iBAQ Before_Induction_01	2.083660	Before_Induction	01
	C0SP93	AccD	iBAQ Before_Induction_01	2.312376	Before_Induction	01
	C0SP94	YhfQ	iBAQ Before_Induction_01	-2.030633	Before_Induction	01
	C0SPA7	YukB	iBAQ Before_Induction_01	-8.627621	Before_Induction	01
	C0SPB0	YtcI	iBAQ Before_Induction_01	0.498673	Before_Induction	01