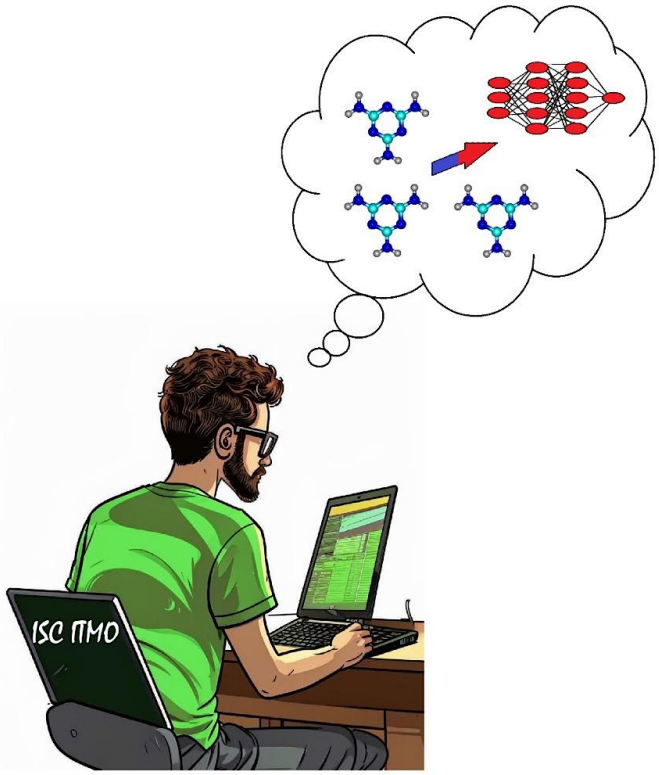
**S. Normatov, P.V. Nesterov, T.A. Aliev,
A.A. Timralieva, A.S. Novikov, E.V. Skorb**

PRACTICE-ORIENTED INTRODUCTION TO MACHINE LEARNING: LINEAR REGRESSION, DECISION TREE, AND SINGLE LAYER PERCEPTRON MODELS



Saint Petersburg

2024

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

**S. Normatov, P.V. Nesterov, T.A. Aliev,
A.A. Timralieva, A.S. Novikov, E.V. Skorb**

PRACTICE-ORIENTED INTRODUCTION TO MACHINE LEARNING: LINEAR REGRESSION, DECISION TREE, AND SINGLE LAYER PERCEPTRON MODELS

УЧЕБНО-МЕТОДИЧЕСКОЕ ПОСОБИЕ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО

по направлению подготовки 04.03.01, 18.03.01

в качестве Учебно-методическое пособие для реализации основных профессиональных образовательных программ высшего образования бакалавриата

ИТМО

Санкт-Петербург

2024

S. Normatov, P.V. Nesterov, T.A. Aliev, A.A. Timralieva, A.S. Novikov, E.V. Skorb, Practice-Oriented Introduction to Machine Learning: Linear Regression, Decision Tree, and Single Layer Perceptron models – СПб: Университет ИТМО, 2024. – 41 с.

Рецензент: Клюкин Илья Николаевич, кандидат химических наук, -, старший научный сотрудник, Институт общей и неорганической химии имени Н. С. Курнакова РАН

This manual aims to provide some basic knowledge on Machine Learning algorithms for the Infochemistry Scientific Center students. It is a practice-oriented tutorial which serves as guideline for the beginners in their desire to build their own Machine Learning models. The Database (DB) used in this work contains the energy values of chemical systems, obtained via Quantum Chemical calculations. However, the protocol mentioned here not only applicable for calculated data, but also for any type of tabular values (including experimental data). The manual covers the main aspects of Machine Learning process, which include exploratory data analysis (EDA), data preprocessing, training, validation, and visualization. The authors provide some examples on three Machine Learning algorithms, namely: Linear Regression, Decision Tree, and Single Layer Perceptron. These three algorithms are considered to be good starting points (in that they are the least complex) in their respective classes: linear, tree-based, and neural networks.

The manual can be useful for teaching Bachelor students of the “Infochemistry” educational program at ITMO University and is within the modules of the following courses: “Infochemistry”, “Artificial Intelligence and Machine Learning in Chemistry”.

Content

INTRODUCTION

Machine Learning is amply applied in the field of Computational Chemistry since it can make instant and accurate predictions of values, which describe the behavior of chemical systems. For example, Quantum Chemical calculations, depending on the level of theory, offer the highest accuracy currently available, but on the other hand they can take from weeks to months to complete even on supercomputers [1,2]. Ab Initio Molecular Dynamics, another type of Computational Chemistry, although considered to be the best technique to model and predict the evolution of chemical / biochemical systems over time, can be actually much more computationally demanding, since the calculations are made for each frame of the simulation [3]. Time is the main factor that limits the application of this resource-intensive computational techniques. Machine Learning algorithms not only have advanced the Computational Chemistry techniques in terms of time (MLIP [4], Δ-DFT [5] etc.), but also in terms of system design. AlphaFold, currently recognized as the best representative of this kind, is AI-based software that accurately predicts the three-dimensional structures of proteins based solely on 1their amino acid sequence [6]. The Royal Swedish Academy of Sciences awarded the Nobel Prize in Chemistry 2024 to the authors of the AlphaFold "for computational protein design" and "for protein structure prediction" [7]. All the above advances highlight the importance and the promising use of Machine Learning algorithms in the field of Computational Chemistry.

This manual aims to provide some basic knowledge on Machine Learning algorithms for the ITMO University students. It is a practice-oriented tutorial which serves as a guideline for the beginners in their desire to build their own Machine Learning models. Hence, readers are going to address multiple code snippets and plots across the manual, for which authors will provide detailed explanations in the corresponding sections. Besides, files with the codes have been properly structured and documented, which would make the learning process more comfortable and intuitively clear to comprehend. For educational purposes, the authors favored using Jupyter Notebooks (".ipynb") over standard Python files (".py") within Visual Studio Code (VSCode) due to the flexibility of the former and their ability to make document-like notations. Visual Studio Code is a versatile programming tool available for Windows, Linux and MacOS users at https://code.visualstudio.com/Download.

The Database (DB) used in this work contains the energy values of chemical systems, obtained via Quantum Chemical calculations. However, the protocol mentioned here not only applicable for calculated data, but also for any type of tabular values (including experimental data). The manual covers the main aspects of the Machine Learning process, which include exploratory data analysis (EDA),

data preprocessing, training, validation, and visualization. The authors provide some examples on three Machine Learning algorithms, namely: Linear Regression, Decision Tree, and Single Layer Perceptron. These three algorithms are considered to be good starting points (in that they are the least complex) in their respective classes [8-10]: linear, tree-based, and neural networks. The code lines for Exploratory Data Analysis and Machine Learning models along with the database in a csv file freely available in GitHub repository at https://github.com/Saadiallakh/Manual_for_ISC_ITMO.

The authors assume the readers to have some basic skills of the Python programming language (at least with its syntax), which is the main prerequisite for this tutorial. The authors also encourage the users to consult the official Python [11], Scikit-Learn [12,13], and TensorFlow [14,15] web-sites to get an idea of the recent updates on the programming language and the ML-specific libraries. To reproduce the workflow of this manual, the readers are recommended to have all the necessary libraries installed to keep them consistent with the versions, listed in the "requirements.txt" file in the GitHub repository.

The manual can be useful for teaching Bachelor students of the "Infochemistry" educational program at ITMO University and is within the modules of the following courses: "Infochemistry", "Artificial Intelligence and Machine Learning in Chemistry".

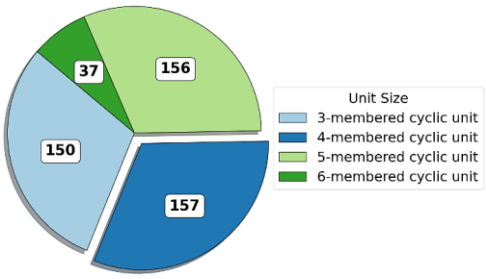
1 DATABASE

"Data.csv" file is a dataset that contains the energy values of dimers, calculated using two quantum chemical approximations (Table 1). It includes 500 observations and 8 features, each of the latter representing one of the 4 molecular descriptors: Gibbs Energy, Electronic Energy, Enthalpy and Entropy. Dimers were generated manually from isolated molecules, selected from the QM9 database [16,17]. All of them consist of non-conjugated cyclic and heterocyclic molecules with a neutral charge. The molecules feature a planar arrangement of rings parallel to each other, with their sizes ranging from 3 to 6 members (Fig. 1A). Heterocyclic dimers contain only two types of heteroatoms: oxygen and nitrogen (Fig. 1B).

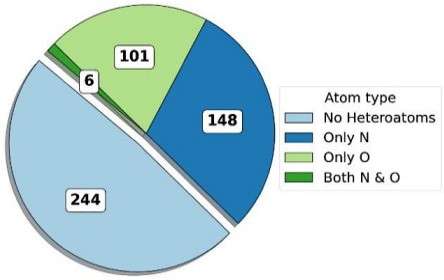
Table 1

Feature types: The "**hf_**" (Hartree-Fock) set was calculated using fast but inaccurate approximation, while "**dft_**" (Density Functional Theory) set was calculated using accurate yet resource-intensive approximation.

Feature	Type	Level of theory
dft_gibbs_free_energy_ev	Target	Gibbs free energy calculated using DFT
dft_electronic_energy_ev	Target	Electronic energy calculated using DFT
dft_entropy_ev	Target	Entropy calculated using DFT
dft_enthalpy_ev	Target	Enthalpy calculated using DFT
hf_gibbs_free_energy_ev	Training	Gibbs free energy calculated using HF
hf_electronic_energy_ev	Training	Electronic energy calculated using HF
hf_entropy_ev	Training	Entropy calculated using HF
hf_enthalpy_ev	Training	Enthalpy calculated using HF



A) Number of Systems by Unit Size



B) Number of Systems by Heteroatoms in the ring

Figure 1. A) Prevalence of systems with 4-membered cyclic dimers; B) Prevalence of systems with non-heterocyclic dimers

2 EXPLORATORY DATA ANALYSIS

Exploratory Data Analysis (EDA) is a crucial phase in the data analysis process, focusing on understanding the dataset's structure, identifying patterns, and preparing the data for further analysis. EDA employs various techniques, including data visualization, summary statistics, and data cleaning, to uncover the insights and inform the subsequent analytical steps. It is an iterative process that helps analysts make informed decisions about the data and the appropriate statistical methods to apply. Statistics covered in this manual are given in Fig. 2. The key aspects of EDA are as follows:

• Data quality assessment: EDA involves checking for errors, outliers, and missing data, which are all crucial for ensuring the dataset's integrity [18,19].

• Descriptive statistics: Calculating summary statistics provides a basic understanding of the data's central tendency, dispersion, and distribution [20,21].

• Data visualization: Visual tools including plots and graphs are used to identify patterns, trends, and relationships within the data, making complex datasets more comprehensible [22,23].

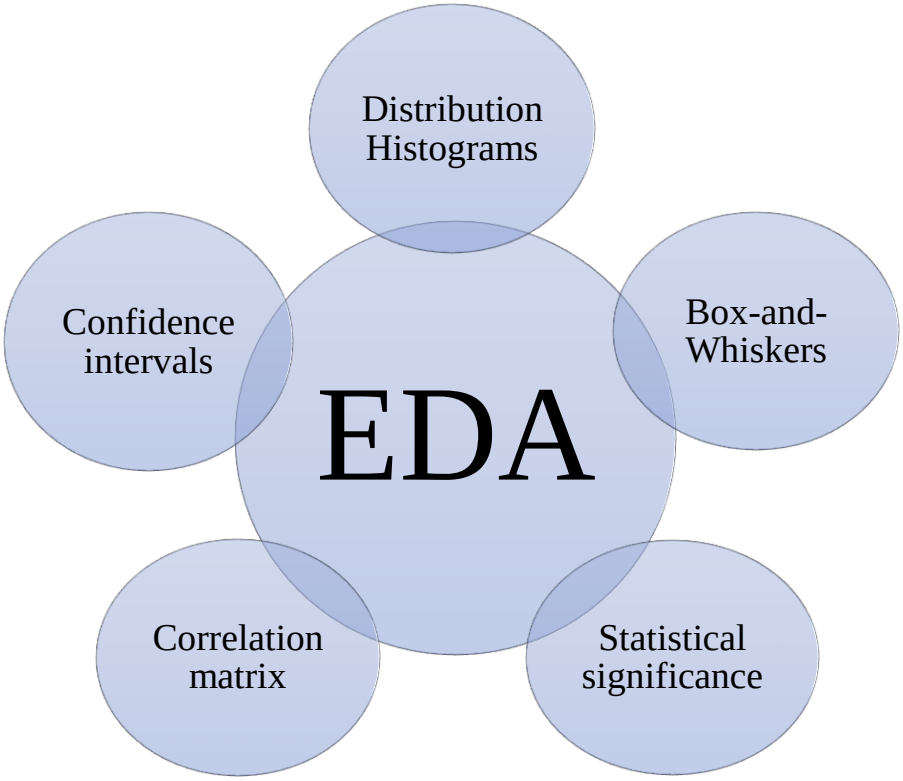


Figure 2. Statistics calculated for dataset analysis

1) Correlation Matrix in Machine Learning is a table showing the correlation coefficients between variables. Each cell in the matrix represents the correlation between two variables, providing insights into their linear relationships [24].

2) Distribution Histograms visualize normal, skewed (left or right), bimodal, uniform or other distribution shapes. This visualization aids in understanding how the data are spread and can highlight the outliers or anomalies [25].

3) Box-and-Whiskers diagrams provide a concise summary of data characteristics, including central tendency, dispersion, and outliers, assuming no normal distribution. The outliers are plotted as individual points beyond the whiskers [26,27].

4) Statistical significance (p-value) is a measure used to determine the significance of the results in null hypothesis testing (reject if $p < 0.05$) [28,29].

5) Confidence Intervals provide a range within which a population parameter is expected to lie with a certain level of confidence [30].

2.1 Confidence Intervals

1) First of all, let us create a **"EDA.ipynb"** file and open it in the VSCode code editor. This file will contain all the code lines for EDA as well as the corresponding outputs (plots, graphs). Let us install and import the **"Pandas"** library to read our **"Data.csv"** file. Leave the first line commented (that is, do not delete the "#-#" symbols) if you have already installed **"Pandas"**.

```
#-# !pip install pandas
import pandas as pd

data = pd.read_csv("Data.csv", delimiter=',')
data
```

If you have successfully executed the code, then the file contents will be displayed right beneath the code lines (the so called "output")

	name	dft_gibbs_free_energy_ev	dft_electronic_energy_ev	dft_entropy_ev	dft_enthalpy_ev	hf_gibbs_free_energy_ev
0	dimer_001338	-16917.573481	-16925.578752	1.791468	-16915.782013	-16699.004234
1	dimer_001340	-15834.675653	-15843.326901	1.798428	-15832.877225	-15625.077800
2	dimer_001341	-16916.895225	-16924.923731	1.759009	-16915.136217	-16698.972462
3	dimer_001343	-14897.694694	-14905.933900	1.622494	-14896.072200	-14700.613585
4	dimer_001346	-14897.734585	-14905.936524	1.662438	-14896.072147	-14700.649672
...	...	...	...	...	...	...
495	dimer_006866	-21012.703301	-21021.225137	1.712411	-21010.990890	-20749.312102
496	dimer_006867	-21012.822136	-21021.382126	1.678073	-21011.144062	-20749.476476
497	dimer_006868	-22968.461459	-22975.705901	1.679857	-22966.781601	-22687.402193
498	dimer_006888	-21011.089783	-21019.404453	1.841686	-21009.248097	-20745.883071
499	dimer_006889	-22967.151760	-22974.201440	1.798617	-22965.353142	-22683.998145

500 rows × 9 columns

2) As we can see, our "data" variable contains 500 rows and 9 columns, which indeed reflects the contents of our "**Data.csv**" file. Now we have to install (**"!pip install scipy"**) and import the **"Scipy"** library to calculate the 95% confidence intervals.

```python
import scipy.stats as stats

dft_columns = [col for col in data.columns if col.startswith("dft_")]
confidence_level = 0.95
z_score = stats.norm.ppf(1 - (1 - confidence_level) / 2)

confidence_intervals = {}

for col in dft_columns:
    mean = data[col].mean()
    std_error = data[col].std() / (len(data[col]) ** 0.5)
    margin_error = z_score * std_error
    lower_bound = mean - margin_error
    upper_bound = mean + margin_error
    width = upper_bound - lower_bound
    confidence_intervals[col] = {
        "mean": mean,
        "lower_bound": lower_bound,
        "upper_bound": upper_bound,
        "width": width
    }

confidence_intervals_df = pd.DataFrame(confidence_intervals).T
confidence_intervals_df
```

CODE ANALYSIS

1. Selecting columns

• **"dft_columns"** filters **"data"** columns to include only those that start with **"dft_"**, storing their names in the list **"dft_columns"**.

2. Setting confidence level

• **Lines 4 & 5** define a 95% confidence level, then calculate the corresponding z-score (1.96 for 95%) using **"stats.norm.ppf()"**. This z-score is used in the calculation of the margin of error for each interval.

3. Calculating confidence intervals

• **Line 7** initializes an empty dictionary to store the results.

• **Lines 9-11** loop over each **"dft_"** column, calculate its mean, and determine the standard error (sample standard deviation divided by the square root of the sample size).

4. Margin of error, Bounds, and Width

• **"margin_error"** line calculates the margin of error for each column by multiplying the z-score by the standard error.

• **Lines 13-14** derive the lower and the upper bounds for the 95% confidence interval around the mean.

• **"width"** – width of the confidence interval - is the difference between the upper and the lower bounds.

5. Storing results

• **Lines 16-21** store the mean, the lower bound, the upper bound, and the width for each **"dft_"** column in the **"confidence_intervals"** dictionary.

6. Converting to DataFrame

• **Lines 23-24** convert the dictionary to a DataFrame, making it easier to view and interpret each column's statistics

3) Our code will output the calculated intervals for the target features. Actually, we can use it to calculate the confidence intervals for the **"hf_"** features as well. For this purpose, we simply have to change the **"dft_"** in the 3rd column to the **"hf_"**.

	mean	lower_bound	upper_bound	width
dft_gibbs_free_energy_ev	-18476.120650	-18632.717676	-18319.523624	313.194053
dft_electronic_energy_ev	-18484.279583	-18640.847777	-18327.711390	313.136387
dft_entropy_ev	1.777621	1.771725	1.783516	0.011791
dft_enthalpy_ev	-18474.343029	-18630.938412	-18317.747647	313.190765

	mean	lower_bound	upper_bound	width
hf_gibbs_free_energy_ev	-18238.798919	-18393.704638	-18083.893201	309.811437
hf_electronic_energy_ev	-18248.489629	-18403.358525	-18093.620734	309.737792
hf_entropy_ev	1.731084	1.725465	1.736703	0.011238
hf_enthalpy_ev	-18237.067836	-18391.971986	-18082.163685	309.808301

Finally, we get our Confidence Intervals for both sets: HF (**“hf_”**) and DFT (**“dft_”**).

2.2 Distribution Histograms

1) We have to install and import the **“Matplotlib”** and **“Seaborn”** packages to plot the Distribution Histograms for our dataset.

```
import matplotlib.pyplot as plt
import seaborn as sns

dft_columns = [
    "dft_gibbs_free_energy_ev",
    "dft_electronic_energy_ev",
    "dft_entropy_ev",
    "dft_enthalpy_ev"
]
titles = {
    "dft_gibbs_free_energy_ev": "(DFT) Gibbs Free Energy",
    "dft_electronic_energy_ev": "(DFT) Electronic Energy",
    "dft_entropy_ev": "(DFT) Entropy",
    "dft_enthalpy_ev": "(DFT) Enthalpy"
}
plt.figure(figsize=(12, 8))
num_columns = len(dft_columns)
rows = (num_columns // 2) + (num_columns % 2)
for i, col in enumerate(dft_columns, 1):
    plt.subplot(rows, 2, i)
    sns.histplot(data[col], kde=True, bins=30, color="green")
    plt.title(titles[col])  #-# Use the custom titles
    plt.xlabel("Energy, eV")
    plt.ylabel("Frequency")
plt.tight_layout()
plt.show()
```

CODE ANALYSIS

1. Import libraries

2. Define columns and titles

• **"dft_columns"** specifies a list of column names in **"data"** that will be plotted

• **"titles"** is a dictionary that assigns custom titles for each **"dft_"** column

3. Setting up the plotting grid

• **Line 16** creates a grid layout to display all histograms in a single figure

• **"figsize"** sets the overall size of the figure

• **"rows"** determines the number of rows in the grid layout for the subplots, ensuring there are enough rows to fit all columns

4. Looping and plotting each histogram

• Loop iterates through each column in **"dft_columns"**, creating a subplot for each

• **Line 22** sets the title for each subplot using **"titles[col]"**

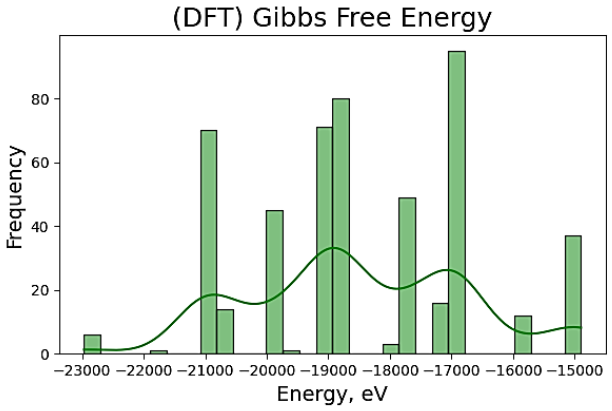
• **Lines 23 & 24** set the x-axis label to the "Energy, eV" and y-axis label to "Frequency"

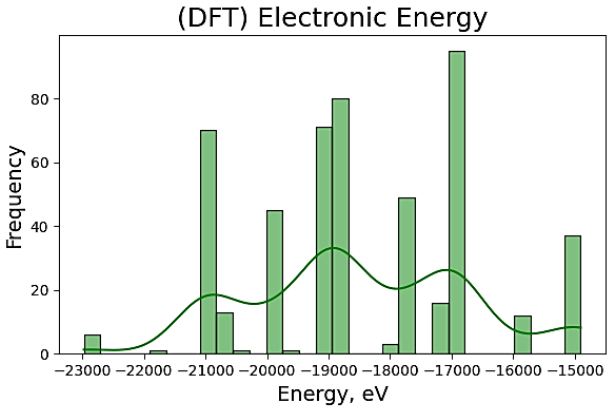
5. Adjust layout and display

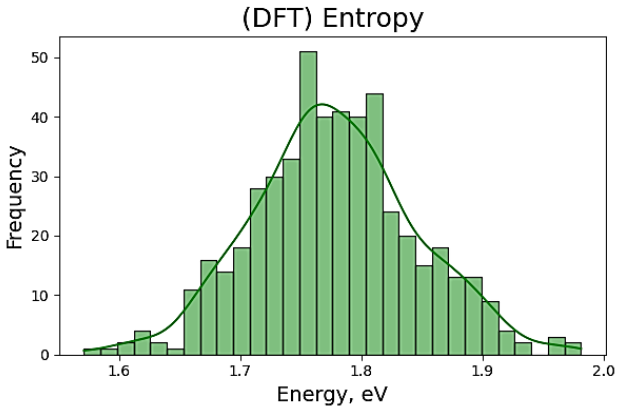
• **"plt.tight_layout()"** adjusts the layout to avoid overlap between subplots

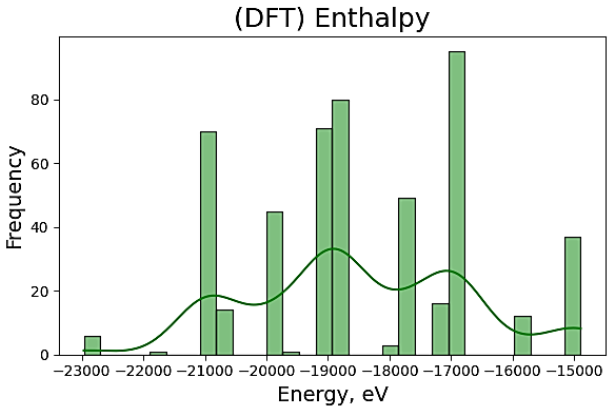
• **"plt.show()"** displays the plot

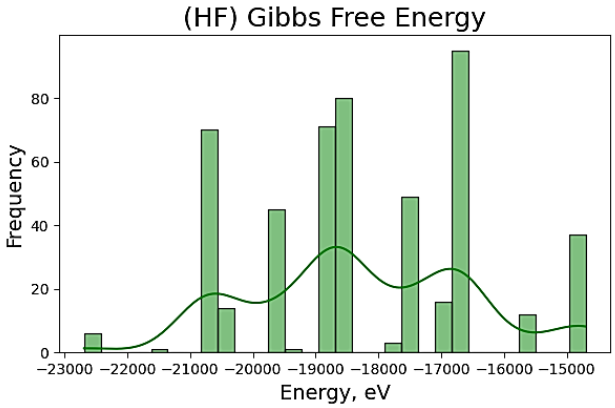
2) The same code lines with some small adjustments can be used to plot the Distribution Histograms for the **"hf_"** values. As a result, you will get a plot with 4 subplots illustrating the distribution of values in each set.

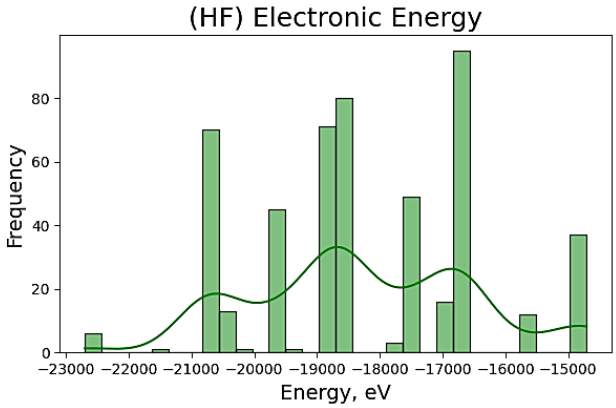

(DFT) Gibbs Free Energy
Frequency
Energy, eV

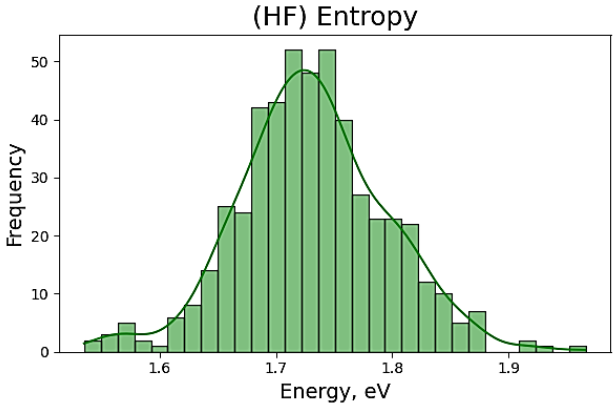

(DFT) Electronic Energy
Frequency
Energy, eV

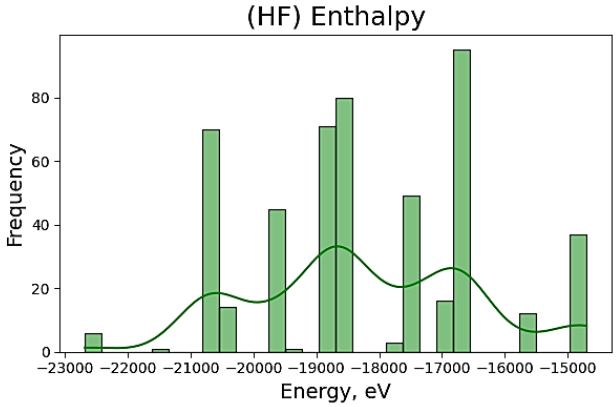

(DFT) Entropy
Frequency
Energy, eV


(DFT) Enthalpy
Frequency
Energy, eV


(HF) Gibbs Free Energy
Frequency
Energy, eV


(HF) Electronic Energy
Frequency
Energy, eV


(HF) Entropy
Frequency
Energy, eV


(HF) Enthalpy
Frequency
Energy, eV

2.3 Correlation Matrix

1) We will use the **“Seaborn”** and **“Matplotlib”** libraries to plot the Correlation Matrix between the **“dft_”** and **“hf_”** sets of values.

```
from matplotlib.colors import LinearSegmentedColormap

dft_columns = [col for col in data.columns if col.startswith("dft_")]
hf_columns = [col for col in data.columns if col.startswith("hf_")]
correlation_matrix = data[dft_columns + hf_columns].corr()
correlation_matrix = correlation_matrix.loc[dft_columns, hf_columns]

dft_labels = {
    "dft_gibbs_free_energy_ev": "(DFT) Gibbs Free Energy",
    "dft_electronic_energy_ev": "(DFT) Electronic Energy",
    "dft_entropy_ev": "(DFT) Entropy",
    "dft_enthalpy_ev": "(DFT) Enthalpy"
}
hf_labels = {
    "hf_gibbs_free_energy_ev": "(HF) Gibbs Free Energy",
    "hf_electronic_energy_ev": "(HF) Electronic Energy",
    "hf_entropy_ev": "(HF) Entropy",
    "hf_enthalpy_ev": "(HF) Enthalpy"
}
correlation_matrix.rename(index=dft_labels, columns=hf_labels, inplace=True)
colors = ["blue", "white", "green"]
custom_blue_green = LinearSegmentedColormap.from_list("CustomBlueGreen", colors)
plt.figure(figsize=(10, 8))
heatmap = sns.heatmap(correlation_matrix, annot=True, cmap=custom_blue_green, fmt=".4f", linewidths=0.5,
                      annot_kws={"size": 14}, cbar=True)
plt.xticks(rotation=45, ha='right', fontsize=14)
plt.yticks(fontsize=14)
colorbar = heatmap.collections[0].colorbar
colorbar.ax.tick_params(labelsize=14)
plt.show()
```

CODE ANALYSIS

1. Filter columns

• **“dft_columns”** and **“hf_columns”** are the lists of columns in **“data”** that start with **“dft_”** and **“hf_”**, respectively. These are used to separate the columns for a correlation analysis.

2. Calculate and subset Correlation Matrix

• **“correlation_matrix”** calculates the correlation matrix for the selected columns and then subsets it so that the **“dft_”** columns are on the Y-axis and **“hf_”** are on the X-axis

3. Define custom labels for columns

• **"dft_labels"** and **"hf_labels"** dictionaries are defined to provide more descriptive labels for the **"dft_"** and **"hf_"** columns

4. Custom colors

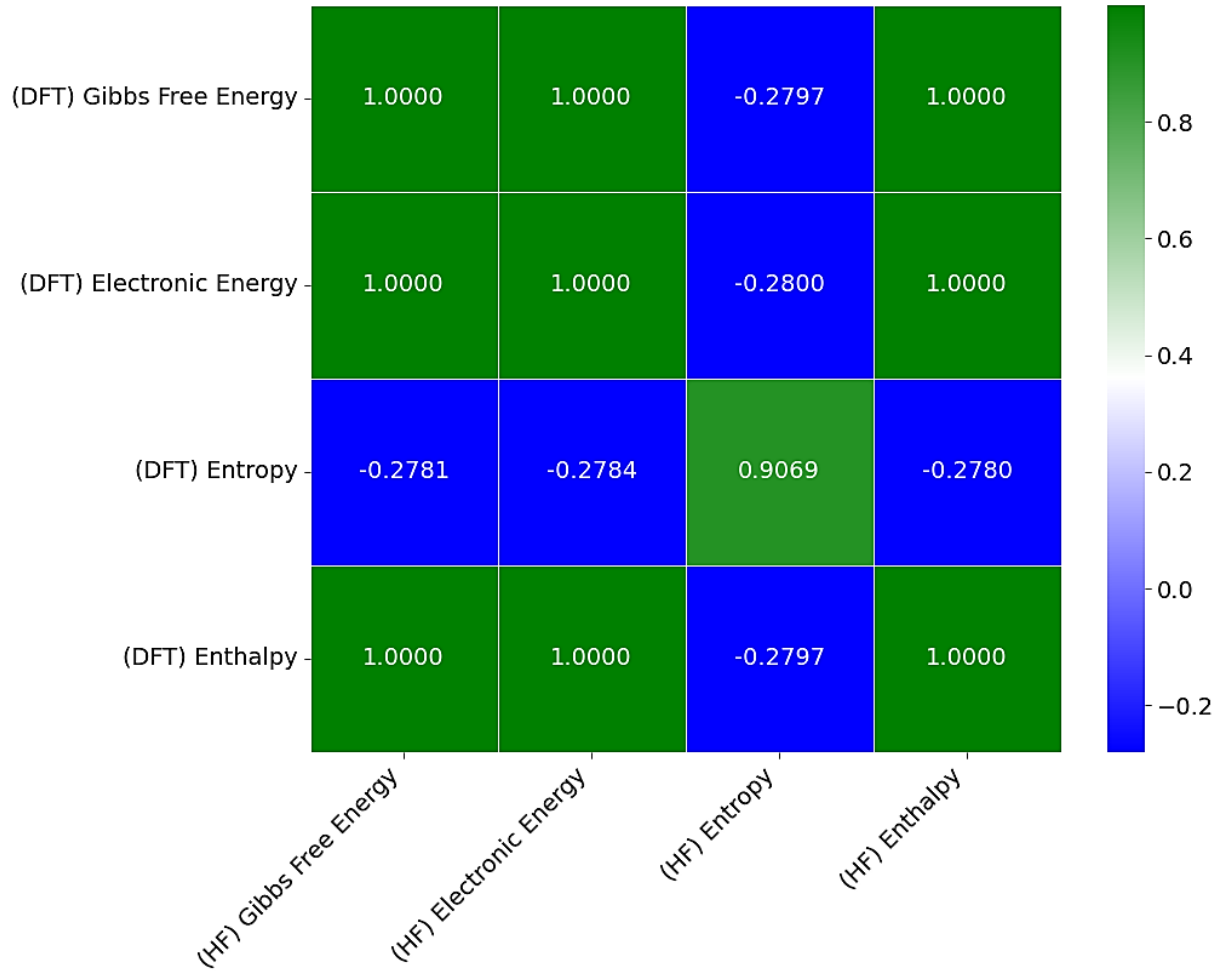
Lines 21 & 22 define color transitions in matrix

5. Plot the Correlation Matrix with heatmap

• **"heatmap"** creates a heatmap with **"sns.heatmap()"**.

• **Lines 26 & 29** set the text parameters for the plot.

2) As a result, you get a 4×4 Correlation Matrix with the color scheme as follows: blue color for the negative, green for the positive and white as a neutral midpoint.



2.4 Statistical Significance (P-values)

1) To calculate the p-values between the two sets of columns, we can use **Pandas** for a quick and straightforward calculation. Additionally, we have to import the **"pearsonr"** from **"scipy.stats"**

```
1  from scipy.stats import pearsonr
2
3  dft_columns = [col for col in data.columns if col.startswith("dft_")]
4  hf_columns = [col for col in data.columns if col.startswith("hf_")]
5
6  p_values = pd.DataFrame(index=dft_columns, columns=hf_columns)
7
8  for hf in hf_columns:
9      for dft in dft_columns:
10         _, p_value = pearsonr(data[hf], data[dft])
11         p_values.loc[dft, hf] = p_value
12
13 p_values
```

CODE ANALYSIS

1. Select columns

• **"dft_columns"** is a list of columns in **"data"** that begin with **"dft_"**, indicating the features related to DFT (Density Functional Theory)

• **"hf_columns"** is a list of columns in **"data"** that begin with **"hf_"**, indicating the features related to HF (Hartree-Fock)

2. Create an empty DataFrame for the p-values

• **"p_values"** is an empty DataFrame with the rows labeled by **"dft_columns"** and the columns labeled by **"hf_columns"**. This structure will store the p-values for each pairwise correlation between the **"dft_"** and the **"hf_"** columns

3. Calculate p-values

• For each combination of a **"dft_"** column and **"hf_"** column, the code calculates the p-value related to their Pearson correlation using **"pearsonr"**.

• **"pearson(data[hf], data[dft])"** returns the p-value **"p_value"**, which represents the statistical significance.

2) Code lines for p-values produce the following output

	hf_gibbs_free_energy_ev	hf_electronic_energy_ev	hf_entropy_ev	hf_enthalpy_ev
dft_gibbs_free_energy_ev	0.0	0.0	0.0	0.0
dft_electronic_energy_ev	0.0	0.0	0.0	0.0
dft_entropy_ev	0.0	0.0	0.0	0.0
dft_enthalpy_ev	0.0	0.0	0.0	0.0

2.5 Box-and-Whiskers

1) To plot the Box-and-Whisker diagrams we use the **"Matplotlib"** and **"Seaborn"** libraries. Additionally, we import **"Math"** to properly arrange the plots

```
import math

dft_columns = [col for col in data.columns if col.startswith("dft_")]
dft_labels = {
    "dft_gibbs_free_energy_ev": "(DFT) Gibbs Free Energy",
    "dft_electronic_energy_ev": "(DFT) Electronic Energy",
    "dft_entropy_ev": "(DFT) Entropy",
    "dft_enthalpy_ev": "(DFT) Enthalpy"
}
n_rows = math.ceil(len(dft_columns) / 2)
sns.set_theme(style="whitegrid", palette="muted", font_scale=1.2)
plt.figure(figsize=(15, n_rows * 5))

for i, col in enumerate(dft_columns):
    plt.subplot(n_rows, 2, i + 1)
    sns.boxplot(
        y=data[col],
        color='dodgerblue',
        linewidth=2.5,
        width=0.3,
        flierprops=dict(marker='o', color='red', markersize=5)
    )
    plt.title(dft_labels.get(col, col), fontsize=18, fontweight='bold')
    plt.ylabel("Energy, eV", fontsize=14)
    plt.grid(False)

plt.tight_layout()
plt.show()
```

CODE ANALYSIS

1. Column selection

• **Line 3** selects columns in **"data"** that begin with **"dft_"**, storing them in the **"dft_columns"** variable

2. Custom titles

• **"dft_labels"** is a dictionary that maps the **"dft_"** column names to custom titles, which are used as subplot titles for clarity

3. Rows calculation

• **"n_rows"** calculates the number of rows needed to fit all the box plots in a grid of 2 plots per row

4. Plot settings

• **Line 11** sets a white grid background, a muted color palette, and a font scaling for the plot

• **Line 12** defines the figure size based on the number of rows needed, with a fixed width of 15 units and a height proportional to **"n_rows"**

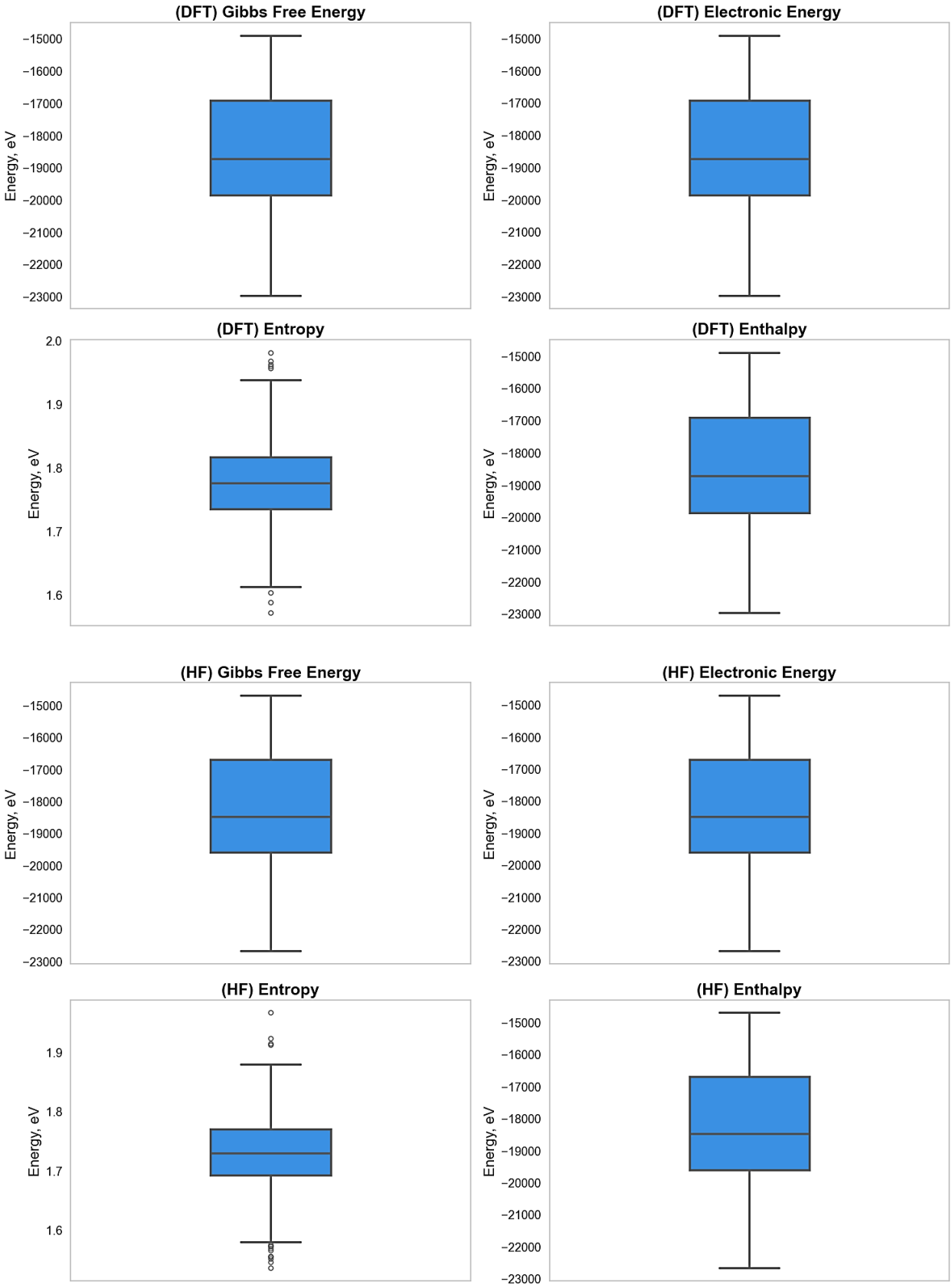
5. Plotting each column in a 2-column grid

• **Line 14** iterates through each **"dft_"** column

• **Line 15** creates a subplot with **"n_rows"** rows and 2 columns, positioning each box plot in the grid

• **"sns.boxplot()"** creates the box plot for each **"dft_"** column with the specified parameters

2) The same code lines can be used to plot the Box-and-Whiskers diagrams for the training set of values. The code outputs should look like plots illustrated below


(DFT) Gibbs Free Energy
(DFT) Electronic Energy
(DFT) Entropy
(DFT) Enthalpy
(HF) Gibbs Free Energy
(HF) Electronic Energy
(HF) Entropy
(HF) Enthalpy
Energy, eV
−15000
−16000
−17000
−18000
−19000
−20000
−21000
−22000
−23000
2.0
1.9
1.8
1.7
1.6

2.6 EDA: Results

1) Distribution histograms

Generally, DFT and HF methods produce highly similar distributions across all properties, suggesting that both methods are in agreement on the general energetic and entropic characteristics of the system. The minor differences in the entropy and slight distribution widths suggest that HF might predict slightly more constrained states in terms of entropy, while DFT may allow for a somewhat higher variability.

2) Box-and-Whiskers

Overall, the DFT calculations show less variability and tend to have lower median values for most energy metrics compared to HF. This implies the DFT might offer more consistent and potentially more stable results. The HF, on the other hand, exhibits a wider range and more outliers, indicating a greater level of fluctuation in its calculations.

3) Correlation Matrix

The DFT and the HF methods agree on entropy calculations but diverge more on other properties. Within each method, properties are more consistently correlated.

4) Confidence Intervals

The DFT and the HF provide close but systematically offset energy predictions, while the entropy estimates are highly consistent between the methods. This consistency and the confidence interval widths suggest reliability in both methods but with a systematic preference for HF to predict slightly less negative energies.

5) Statistical Significance (P-values)

The results confirm that the systematic differences observed in the energy and entropy values between the DFT and the HF are statistically robust. This supports a consistent trend of HF predicting slightly higher (less negative) energies and slightly lower entropy compared to DFT, suggesting that the DFT and the HF yield meaningfully different results, which is likely to result from intrinsic methodological differences rather than from random variation.

3 MACHINE LEARNING

After getting some insights into to the statistics of our database, we can move on to building Machine Learning models. In this manual we cover three algorithms: Linear Regression, Decision Tree, and Single Layer Perceptron. This section of the manual includes three subsections: Data Preprocessing and Splitting, Model Building and Visualization.

3.1 Data Preprocessing and Splitting

1) First of all, let us create a **"Preprocessing.ipynb"** file to start the data preparation. As usual, we need to import the **"Pandas"** library and read our **"Data.csv"** file.

```
#-# !pip install pandas
import pandas as pd

data = pd.read_csv("Data.csv", delimiter=',')
data
```

2) Next, we specify the data that will be used to train the model along with the data for testing the predictive accuracy of the model. We will use the **"hf_"** set of values **(X)** to train our model and the **"dft_enthalpy_ev"** as the target **(y)**.

```
X = data[['hf_gibbs_free_energy_ev',
          'hf_electronic_energy_ev',
          'hf_entropy_ev', 'hf_enthalpy_ev']]

y = data['dft_enthalpy_ev']
```

3) Now we are ready to split our dataset into the training and testing sets. We allocate 80% of data for training and 20% for testing. For this purpose, we import **"train_test_split"** from the **"Sklearn"** library

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

4) After that we normalize our data using **StandardScaler()**. This process is crucial for ensuring that each feature contributes equally to the analysis [31,32]. Generally, it is a good idea to save the scalers to have a better future control on the models.

```
from sklearn.preprocessing import StandardScaler
import joblib

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

joblib.dump(scaler, 'scaler.pkl')
```

3.2 Model Building and Visualization

For the sake of simplicity, we will not mention the preprocessing steps again in this section. These steps are consistent across all the three models and are included in the respective ".ipynb" files. Note that the preprocessing code must be executed simultaneously with the model code.

RMSE, MAE, R^2 and MSE are the accuracy metrics that we will use to evaluate the performance of our models. Root Mean Squared Error (RMSE) measures the average magnitude of the error between the predicted and the observed values, providing a single value that represents the model's prediction error [33,34]. Mean Absolute Error (MAE) measures the average magnitude of the errors in a set of predictions, without considering their direction [35,36]. R^2, or the coefficient of determination, is a statistical measure in Machine Learning that quantifies the proportion of variance in the dependent variable that is predictable from independent variables [37-39]. Mean Squared Error (MSE) measures the average of the squares of the errors, that is, the average squared difference between the estimated values and the actual values [40].

3.2.1 Linear Regression

1) Now we can make a copy of the **"Preprocessing.ipynb"** and rename it to the **"LinearRegression.ipynb"**. We have to import **"LinearRegression"** from the **"Sklearn"** library to build our model. After that we train our model on the training set.

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()
model.fit(X_train_scaled, y_train)
```

2) Let us save the model weights to be able to later make predictions without training process

```
joblib.dump(model, 'linear_regression_model.pkl')
```

3) Make predictions on the test set and calculate the evaluation metrics

```
import numpy as np
from sklearn.metrics import mean_absolute_error, r2_score
from sklearn.metrics import mean_squared_error

y_pred = model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

4) The following results were obtained for the Linear Regression model

- MSE: 2.7446
- RMSE: 1.6567
- MAE: 1.4099
- R^2: 1.0000

5) Visualize and save the results

```
1  import matplotlib.pyplot as plt
2
3  plt.figure(figsize=(10, 6))
4  plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
5  plt.xlabel("Actual", fontsize=18)
6  plt.ylabel("Predicted", fontsize=18)
7  plt.title("Enthalpy, eV", fontsize=22)
8  plt.grid(False)
9
10 plt.xticks(fontsize=14)
11 plt.yticks(fontsize=14)
12
13 metrics_text = f"MSE: {mse:.4f}\nRMSE: {rmse:.4f}\nMAE: {mae:.4f}\n$R^2$: {r2:.4f}"
14 plt.text(0.05, 0.95, metrics_text, transform=plt.gca().transAxes, fontsize=16,
15         verticalalignment='top',
16         bbox=dict(boxstyle="round,pad=0.3", edgecolor='gray', facecolor='lightgreen'))
17
18 plt.show()
19
20 results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
21 results.to_csv('Linear_Regression(Act_vs_Pred).csv', index=False)
```

CODE ANALYSIS

1. Set up the plot

• **Line 1** creates a new figure specified with dimensions

• **Line 2** plots a scatter plot of **"y_test"** (actual values) versus **"y_pred"** (predicted values) in blue with 60% transparency (**"alpha=0.6"**)

• **Lines 5 & 6** label the x-axis as "Actual" and the y-axis as "Predicted"

• **"plt.title"** adds a title "Enthalpy, eV" with larger font size

• **Line 8** disables the grid to give the plot a cleaner look

2. Adjust tick sizes

• **Lines 10 & 11** set the font size for both x-axis and y-axis ticks to 14 for consistency and readability

3. Display model metrics on the plot

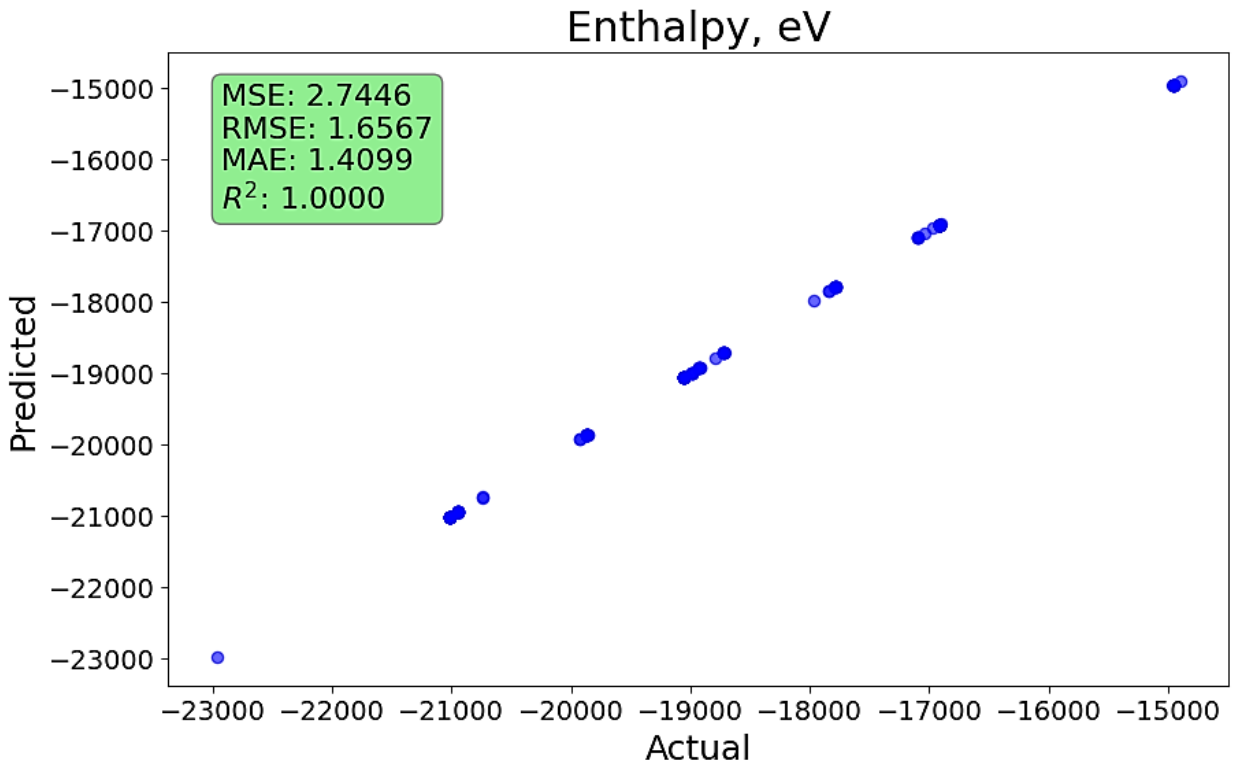
• **“metrics_text”** is a formatted string that displays model evaluation metrics (MSE, RMSE, MAE, R^2), each with four decimal places

• **Lines 14 & 16** add this text to the upper left corner of the plot (**“transform=plt.gca().transAxes”** anchors it relative to the plot area, and to align the text at the top the **“verticalalignment=’top’”** is specified). **“bbox”** – sets light green box surrounding the text for readability (with a gray border).

4. Save results to a csv

• **“results”** contains the DataFrame of Actual and Predicted values, which is subsequently saved as **“Linear_Regression(Act_vs_Pred).csv”** file.

The plot itself with the accuracy metrics looks as follows:



3.2.2 Decision Tree

1) Let us make one more copy of the **“Preprocessing.ipynb”** file and rename it to the **“DecisionTree.ipynb”**. Our models become more complex: now

we have to find the best hyperparameter values to train our model (that is, we have to perform the so called “hyperparameter tuning”).

Hyperparameters in Machine Learning are the crucial elements that define the behavior and the performance of algorithms during the training process. Unlike the model parameters, which are learned from the data, hyperparameters are set before the learning process begins and can significantly influence the model’s ability to generalize to unseen data [41-43]. In this manual, we have given preference to the **Optuna** automatic hyperparameter optimization software framework. **Optuna** is considered superior to **GridSearchCV** for hyperparameter optimization due to its advanced features and its efficiency in exploring the hyperparameter space [44,45].

We optimize the following hyperparameters:

• max_depth

• min_samples_split

• min_samples_leaf

• max_features

All of them are the parameters of **“DecisionTreeRegressor”** of the **“Sklearn”** library, which we will import in our file. The **“max_depth”** controls the maximum depth of the tree. The **“min_samples_split”** specifies the minimum number of samples required to split an internal node. The **“min_samples_leaf”** is a minimum number of samples required to be at a leaf node. The **“max_features”** is the number of features to consider when looking for the best split. In this manual, we skip the theory behind these hyperparameters, thus to get a better idea about them we encourage readers to visit the official web-site of scikit-learn library and go through the **“DecisionTreeRegressor”** page [46].

First, let us import all the necessary libraries

```
1  import optuna
2  import numpy as np
3
4  from sklearn.tree import DecisionTreeRegressor
5  from sklearn.model_selection import KFold
6  from sklearn.metrics import mean_squared_error
```

2) Now we have to create an **“objective”** function where we will specify all the hyperparameters and conditions for their optimization

```python
def objective(trial):

    param = {
        'max_depth': trial.suggest_int('max_depth', 3, 20),
        'min_samples_split': trial.suggest_int('min_samples_split', 2, 20),
        'min_samples_leaf': trial.suggest_int('min_samples_leaf', 1, 20),
        'max_features': trial.suggest_categorical('max_features', ['sqrt', 'log2', 0.5, 0.8, 1.0])
    }

    model = DecisionTreeRegressor(**param, random_state=42)

    kf = KFold(n_splits=3, shuffle=True, random_state=42)
    mse_list = []

    for train_index, val_index in kf.split(X_train_scaled):
        X_train_cv, X_valid_cv = X_train_scaled[train_index], X_train_scaled[val_index]
        y_train_cv, y_valid_cv = y_train.iloc[train_index], y_train.iloc[val_index]

        model.fit(X_train_cv, y_train_cv)

        y_pred_cv = model.predict(X_valid_cv)
        mse = mean_squared_error(y_valid_cv, y_pred_cv)
        mse_list.append(mse)

    return np.mean(mse_list)
```

CODE ANALYSIS

1. Define hyperparameter grid

• **"trial.suggest_*"** functions define the hyperparameter grid, which allows **Optuna** to sample various values during each trial

2. Initialize the DecisionTreeRegressor

• The model is created using **"DecisionTreeRegressor"** with the parameters suggested in the current trial, and **"random_state=42"** for reproducibility.

3. Set up cross-validation

• The **"KFold"** object performs 3-fold cross-validation (splitting the data into 3 parts) and shuffles the data before splitting to ensure varied splits across the trials

4. Run cross-validation

• For each split of the data: 1) Data is divided into the training and validation sets; 2) The model is trained on the training fold (**"model.fit()"**); 3) Predictions are made on the validation fold; 4) MSE is calculated between the

predicted values (**“y_pred_cv”**) and actual values (**“y_valid_cv”**). This MSE is appended to the **“mse_list”**, which stores the MSE for each fold.

5. Return the mean MSE

• Finally, the function returns the mean of the MSE values across all folds.

3) Run the optimization. For the sake of time, we run only 50 trials, but to get more accurate results, you would need to run from 1000 to 5000 trials.

```
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=10)
```

If everything has been done correctly, the code will output the optimization progress

```
[I 2024-11-02 11:41:01,330] A new study created in memory with name: no-name-f117a71d-d2fb-4777-9e42-262c9eedeb4a
[I 2024-11-02 11:41:01,348] Trial 0 finished with value: 20131.27228559589 and parameters: {'max_depth': 5, 'min_sa
[I 2024-11-02 11:41:01,361] Trial 1 finished with value: 60551.271536101725 and parameters: {'max_depth': 9, 'min_s
[I 2024-11-02 11:41:01,371] Trial 2 finished with value: 56219.95888639641 and parameters: {'max_depth': 17, 'min_s
[I 2024-11-02 11:41:01,385] Trial 3 finished with value: 20969.511111748787 and parameters: {'max_depth': 7, 'min_s
[I 2024-11-02 11:41:01,418] Trial 4 finished with value: 34239.87016187225 and parameters: {'max_depth': 11, 'min_s
[I 2024-11-02 11:41:01,434] Trial 5 finished with value: 2632.92125272494 and parameters: {'max_depth': 6, 'min_sam
```

4) Print the best hyperparameters

```
best_trial = study.best_trial
print(f'Best trial value: {best_trial.value}')
print(f'Best hyperparameters: {best_trial.params}')
```

We get the following combination:

• Best trial value: 2068.5881528457116

• Best hyperparameters: {'**max_depth**': 12, '**min_samples_split**': 7, '**min_samples_leaf**': 1, '**max_features**': 0.8}

5) Let us save the optimization results

```
with open('optimization_results.txt', 'w') as f:
    f.write(f'Best trial value: {best_trial.value}\n')
    f.write(f'Best hyperparameters: {best_trial.params}\n')
    f.write('\nAll trial results:\n')
    for trial in study.trials:
        f.write(f'Trial {trial.number}: Value={trial.value}, Params={trial.params}\n')
```

6) Retrieve the best hyperparameters to pass them for model training

```
best_params = study.best_params
```
✓ 0.0s

7) Train the model with the best hyperparameters

```
final_model = DecisionTreeRegressor(**best_params, random_state=42)
final_model.fit(X_train_scaled, y_train)
```

8) Save the model weights

```
joblib.dump(final_model, "decision_tree_model.pkl")
```
✓ 0.0s

9) Make predictions on the test set and calculate the evaluation metrics

```
from sklearn.metrics import mean_absolute_error, r2_score

y_pred = final_model.predict(X_test_scaled)

mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

10) The following results were obtained for the Decision Tree model

- MSE: 56.9403
- RMSE: 7.5459
- MAE: 1.1456
- R^2: 1.0000

11) Visualize and save the results

```
1  import matplotlib.pyplot as plt
2
3  plt.figure(figsize=(10, 6))
4  plt.scatter(y_test, y_pred, color='blue', alpha=0.6)
5  plt.xlabel("Actual", fontsize=18)
6  plt.ylabel("Predicted", fontsize=18)
7  plt.title("Enthalpy, eV", fontsize=22)
8  plt.grid(False)
9
10 plt.xticks(fontsize=14)
11 plt.yticks(fontsize=14)
12
13 metrics_text = f"MSE: {mse:.4f}\nRMSE: {rmse:.4f}\nMAE: {mae:.4f}\n$R^2$: {r2:.4f}"
14 plt.text(0.05, 0.95, metrics_text, transform=plt.gca().transAxes, fontsize=16,
15         verticalalignment='top',
16         bbox=dict(boxstyle="round,pad=0.3", edgecolor='gray', facecolor='lightgreen'))
17
18 plt.show()
19
20 results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
21 results.to_csv('Decision_Tree(Act_vs_Pred).csv', index=False)
```

CODE ANALYSIS

1. Set up the plot

- **Line 1** creates a new figure specified with dimensions
- **Line 2** plots a scatter plot of **"y_test"** (actual values) versus **"y_pred"** (predicted values) in blue with 60% transparency (**"alpha=0.6"**)
- **Lines 5 & 6** label the x-axis as "Actual" and the y-axis as "Predicted"

• **"plt.title"** adds a title "Enthalpy, eV" with a larger font size

• **Line 8** disables the grid to give the plot a cleaner look

2. Adjust tick sizes

• **Lines 10 & 11** set the font size for both the x-axis and y-axis ticks to 14 for consistency and readability

3. Display model metrics on the plot

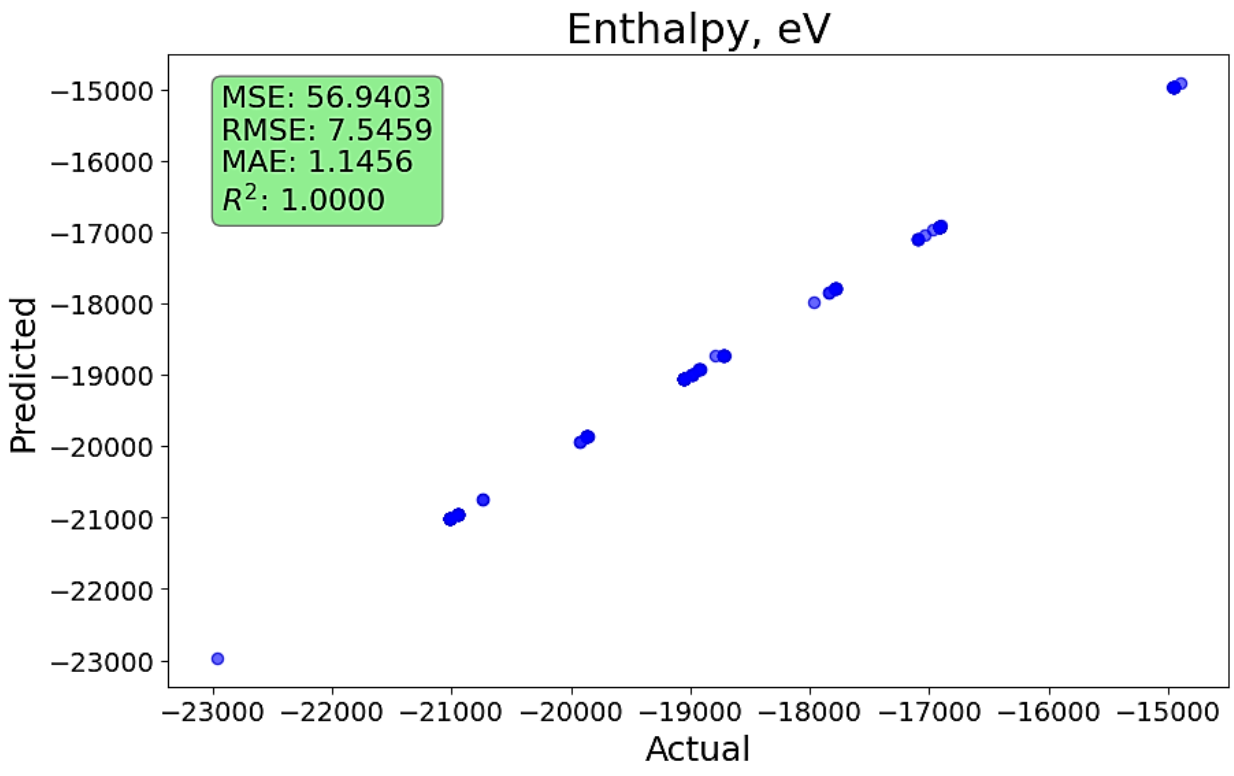
• **"metrics_text"** is a formatted string that displays the model evaluation metrics (MSE, RMSE, MAE, R^2), each with four decimal places

• **Lines 14 & 16** add this text to the upper left corner of the plot (**"transform=plt.gca().transAxes"** anchors it relative to the plot area, and to align the text at the top the **"verticalalignment='top'"** is specified). **"bbox"** – sets light green box surrounding the text for readability (with a gray border).

4. Save results to a csv

• **"results"** contains the DataFrame of Actual and Predicted values, which is subsequently saved as **"Decision_Tree(Act_vs_Pred).csv"** file.

The plot for the Decision Tree with the accuracy metrics looks as follows:



3.2.3 Single Layer Perceptron

Single Layer Perceptron is the simplest form of a neural network, primarily used for linear regression and binary classification tasks. It consists of a single layer of output nodes connected directly to the input features, without any hidden layers. This simplicity makes it a foundational model in neural network research [47].

We will use the **TensorFlow** library to build the model and the **Optuna** library to optimize the following hyperparameters:

- optimizer_name
- activation_function
- learning_rate
- num_neurons
- l1_reg
- l2_reg

Optimizers in Machine Learning are the algorithms designed to adjust the parameters of the models to minimize the loss function, thereby improving the model performance (**"optimizer_name"**). They play a crucial role in training Neural Networks by determining the speed and the quality of convergence [48,49]. Activation functions are critical components in Neural Networks, serving as the decision-making elements that determine the output of a neural node (**"activation_function"**). They introduce non-linearity into the network, enabling it to learn some complex patterns and relationships within the data [50,51]. The **"learning_rate"** is a hyperparameter that determines the step size at each iteration while moving toward a minimum of the loss function [52]. Neurons in Neural Networks are computational units inspired by biological neurons, designed to process and transmit the information through the interconnected layers (**"num_neurons"**). These artificial neurons form the building blocks of Neural Networks, enabling them to perform some complex tasks including pattern recognition and predictive modelling [53]. Regularization in Machine Learning is a technique used to prevent overfitting by introducing some additional information or constraints into the model. It modifies the loss function to include the penalty term, which can help improve the model's generalization to the unseen data [54,55]. Two common forms of regularization are L1 [56,57] and L2 [58,59] regularization, which are implemented the **TensorFlow** or similar frameworks.

1) Let us create a **"SingleLayerPerceptron.ipynb"** file to build our model. This time we need to scale the **"y"** values as well. Single Layer Perceptron, like most Neural Networks, relies on **gradient descent** to iteratively update weights

to minimize the loss function [60-62]. Gradient descent can struggle if the scale of the target variable is too large or too different from the scale of the features. In contrast, Linear Regression and Decision Tree do not depend on **gradient descent**, so they can more easily handle varying scales in the target variable.

Now we have to install and import all the necessary libraries and create the **"objective"** function for **Optuna** optimization, but before doing so let us scale the **"y"** values.

```
from sklearn.preprocessing import StandardScaler
import joblib

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

joblib.dump(scaler, 'scaler.pkl')

scaler_y = StandardScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))
```

2) After this we install and import libraries

```
import optuna
import numpy as np
import tensorflow as tf

from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.regularizers import l1_l2
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error
```

3) Create a function for optimization

```
def objective(trial):
    optimizer_name = trial.suggest_categorical('optimizer', ['Adam', 'SGD', 'RMSprop', 'Adagrad'])
    activation_function = trial.suggest_categorical('activation_function', ['relu', 'sigmoid', 'tanh', 'leaky_relu'])
    learning_rate = trial.suggest_float('learning_rate', 1e-5, 1e-2, log=True)
    num_neurons = trial.suggest_int('num_neurons', 5, 200, log=True)
    l1_reg = trial.suggest_float('l1_reg', 1e-8, 100, log=True)
    l2_reg = trial.suggest_float('l2_reg', 1e-8, 100, log=True)
```

```
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(X_train_scaled.shape[1],)),
        tf.keras.layers.Dense(num_neurons, activation=activation_function, kernel_regularizer=l1_l2(l1=l1_reg, l2=l2_reg)),
        tf.keras.layers.Dense(1)
    ])
```

```
    optimizer = getattr(tf.keras.optimizers, optimizer_name)(learning_rate=learning_rate)
    model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mse'])
    kf = KFold(n_splits=3, shuffle=True, random_state=42)
    fold_mses = []
```

```
for train_index, val_index in kf.split(X_train_scaled):
    X_train_cv, X_val_cv = X_train_scaled[train_index], X_train_scaled[val_index]
    y_train_cv, y_val_cv = y_train_scaled[train_index], y_train_scaled[val_index]
    early_stopping = EarlyStopping(monitor='val_loss', mode='min', restore_best_weights=True, patience=10, verbose=0)
    model.fit(X_train_cv, y_train_cv, epochs=100, batch_size=32, validation_data=(X_val_cv, y_val_cv), callbacks=[early_stopping], verbose=0)
    y_pred_cv = model.predict(X_val_cv)
    mse = mean_squared_error(y_val_cv, y_pred_cv)
    fold_mses.append(mse)
return np.mean(fold_mses)
```

CODE ANALYSIS

1. Objective function for hyperparameter search

• The function **"objective(trial)"** defines the task of finding the best combination of hyperparameters

• It uses **Optuna's "trial"** object to sample various hyperparameter values

• The function returns the MSE across multiple folds of cross-validation, aiming to minimize this value

2. Model definition with current hyperparameters

• A simple feedforward neural network is created with: an input layer matching the shape of the input data; a single hidden layer with **"num_neurons"** and the chosen **"activation_function"**; a regularization function (L1 and L2 regularization); an output layer with one neuron for regression.

3. Optimizer selection

• The optimizer is chosen based on the **"optimizer_name"**, and configured with the suggested **"learning_rate"**.

4. Model compilation

• The model is compiled with: the chosen **"optimizer"** and Mean Squared Error (**"mean_squared_error"**) as the loss function.

5. Cross-validation setup with KFold

• The function uses 3-fold cross-validation (**"KFold(n_splits=3"**)) to ensure the model's performance is tested on different subsets of the data.

4) Run the optimization. For the sake of time, we run only 50 trials, but to get more accurate results, you would need to run from 1000 to 5000 trials.

```
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)
```

If everything has been done correctly, the code will output the optimization progress

```
[I 2024-11-02 23:29:09,595] A new study created in memory with name: no-name-3b357043-4a05-42a8-a898-3ac151ff2b55
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
[I 2024-11-02 23:29:35,851] Trial 0 finished with value: 0.004717398814846414 and parameters: {'optimizer': 'Adam', '
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 10ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
[I 2024-11-02 23:30:01,207] Trial 1 finished with value: 0.5991268591777745 and parameters: {'optimizer': 'Adam', 'ac
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

5) Print the best hyperparameters

```
best_trial = study.best_trial
print(f'Best trial value: {best_trial.value}')
print(f'Best hyperparameters: {best_trial.params}')
```

We get the following combination:

• Best trial value: 2.221684143808076e-06

• Best hyperparameters: {'**optimizer**': 'RMSprop', '**activation_function**': 'leaky_relu', '**learning_rate**': 0.00020565507486431957, '**num_neurons**': 199, '**l1_reg**': 2.3245876147791472e-05, '**l2_reg**': 4.371907855050314e-08}

6) Save the optimization results

```
with open('optimization_results.txt', 'w') as f:
    f.write(f'Best trial value: {best_trial.value}\n')
    f.write(f'Best hyperparameters: {best_trial.params}\n')

    f.write('\nAll trial results:\n')
    for trial in study.trials:
        f.write(f'Trial {trial.number}: Value={trial.value}, Params={trial.params}\n')
```

7) Retrieve the best hyperparameters to pass them for model training

```
best_params = study.best_params
```
✓ 0.0s

8) Build and train the final model with the best hyperparameters

```
model = tf.keras.Sequential([
    tf.keras.layers.Input(shape=(X_train_scaled.shape[1],)),
    tf.keras.layers.Dense(best_params['num_neurons'],
                          activation=best_params['activation_function'],
                          kernel_regularizer=l1_l2(l1=best_params['l1_reg'],
                                                   l2=best_params['l2_reg'])),
    tf.keras.layers.Dense(1)
])
```

9) Choose the best optimizer and implement **“EarlyStopping”** to prevent overfitting. After this compile the model

```
optimizer = getattr(tf.keras.optimizers, best_params['optimizer'])(learning_rate=best_params['learning_rate'])
early_stopping = EarlyStopping(monitor='val_loss', mode='min', restore_best_weights=True, patience=10, verbose=0)
model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mse'])
```

10) Train the model with the best hyperparameters

```
history = model.fit(X_train_scaled, y_train_scaled,
                    epochs=100, batch_size=32,
                    validation_split=0.1, callbacks=[early_stopping], verbose=1)
```

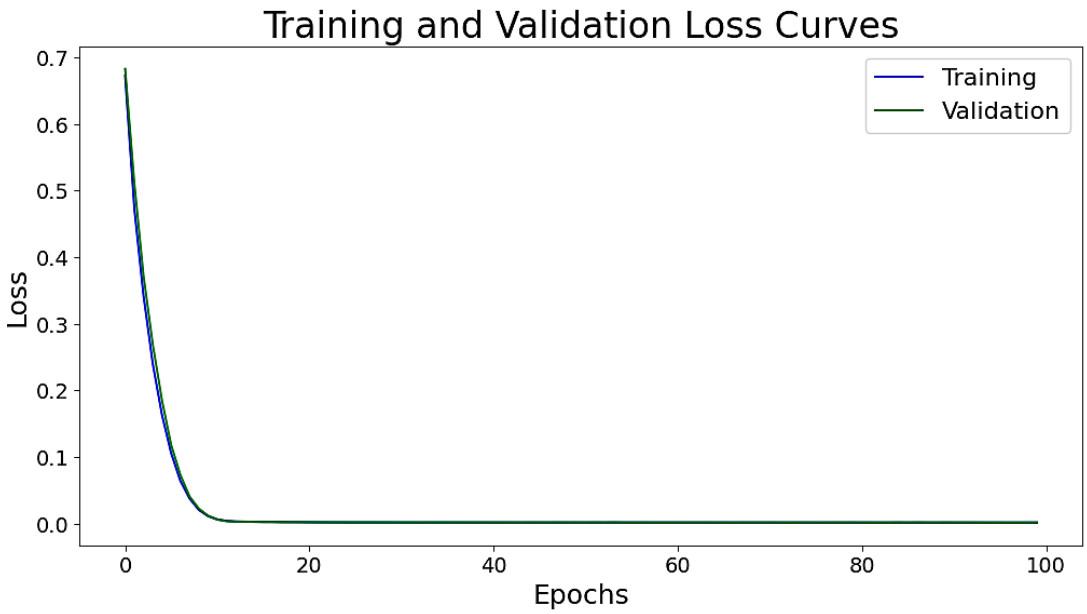
11) Save the training and validation loss to the .txt file

```
with open('training_validation_loss.txt', 'w') as f:
    f.write("Epoch\tTraining Loss\tValidation Loss\n")
    for epoch, (train_loss, val_loss) in enumerate(zip(history.history['loss'], history.history['val_loss']), 1):
        f.write(f"{epoch}\t{train_loss:.4f}\t{val_loss:.4f}\n")
print("Training and validation loss curves saved to ./training_validation_loss.txt")
```

12) Plot the training and validation curves

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))
plt.plot(history.history['loss'], label='Training', color='blue')
plt.plot(history.history['val_loss'], label='Validation', color='green')

plt.xlabel('Epochs', fontsize=18)
plt.ylabel('Loss', fontsize=18)

plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

plt.title('Training and Validation Loss Curves', fontsize=24)
plt.legend(fontsize=16)
plt.grid(False)

plt.savefig('training_validation_loss.png', bbox_inches='tight')
print("Plot saved to ./training_validation_loss.png")
```

You should get the following plot



13) Save the model weights

```
model.save('slp_model.keras')
print("Model saved to ./slp_model.keras")
```

14) Make predictions on the test set

```
y_pred = model.predict(X_test_scaled).flatten()
```
✓ 0.1s

15) Inverse the scaled data

```
y_pred_scaled_reshaped = y_pred.reshape(-1, 1)
y_test_scaled_reshaped = y_test_scaled.reshape(-1, 1)

y_pred_inverse = scaler_y.inverse_transform(y_pred_scaled_reshaped).flatten()
y_test_inverse = scaler_y.inverse_transform(y_test_scaled_reshaped).flatten()
```

16) Calculate the accuracy metrics

```
from sklearn.metrics import mean_absolute_error, r2_score

mse = mean_squared_error(y_test_inverse, y_pred_inverse)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test_inverse, y_pred_inverse)
r2 = r2_score(y_test_inverse, y_pred_inverse)

print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"R^2 Score: {r2:.4f}")
```

17) The following results were obtained for the Single Layer Perceptron model

- MSE: 16.2898
- RMSE: 4.0361
- MAE: 2.8663
- R^2: 1.0000

18) Visualize the results and save the “Actual VS Predicted” values in csv format

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 6))
plt.scatter(y_test_inverse, y_pred_inverse, color='blue', alpha=0.6)
plt.xlabel("Actual", fontsize=18)
plt.ylabel("Predicted", fontsize=18)
plt.title("Enthalpy, eV", fontsize=22)
plt.grid(False)

plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

metrics_text = f"MSE: {mse:.4f}\nRMSE: {rmse:.4f}\nMAE: {mae:.4f}\n$R^2$: {r2:.4f}"
plt.text(0.05, 0.95, metrics_text, transform=plt.gca().transAxes, fontsize=16,
         verticalalignment='top',
         bbox=dict(boxstyle="round,pad=0.3", edgecolor='gray', facecolor='lightgreen'))

plt.show()

results = pd.DataFrame({'Actual': y_test_inverse, 'Predicted': y_pred_inverse})
results.to_csv('SLP(Act_vs_Pred).csv', index=False)
```

CODE ANALYSIS

1. Set up the plot

• **Line 1** creates a new figure with dimensions specified

• **Line 2** plots a scatter plot of **"y_test"** (actual values) versus **"y_pred"** (predicted values) in blue with 60% transparency (**"alpha=0.6"**)

• **Lines 5 & 6** label the x-axis as "Actual" and the y-axis as "Predicted"

• **"plt.title"** adds the title "Enthalpy, eV" with a larger font size

• **Line 8** disables the grid to give the plot a cleaner look

2. Adjust tick sizes

• **Lines 10 & 11** set the font size for both x-axis and y-axis ticks to 14 for consistency and readability

3. Display model metrics on the plot

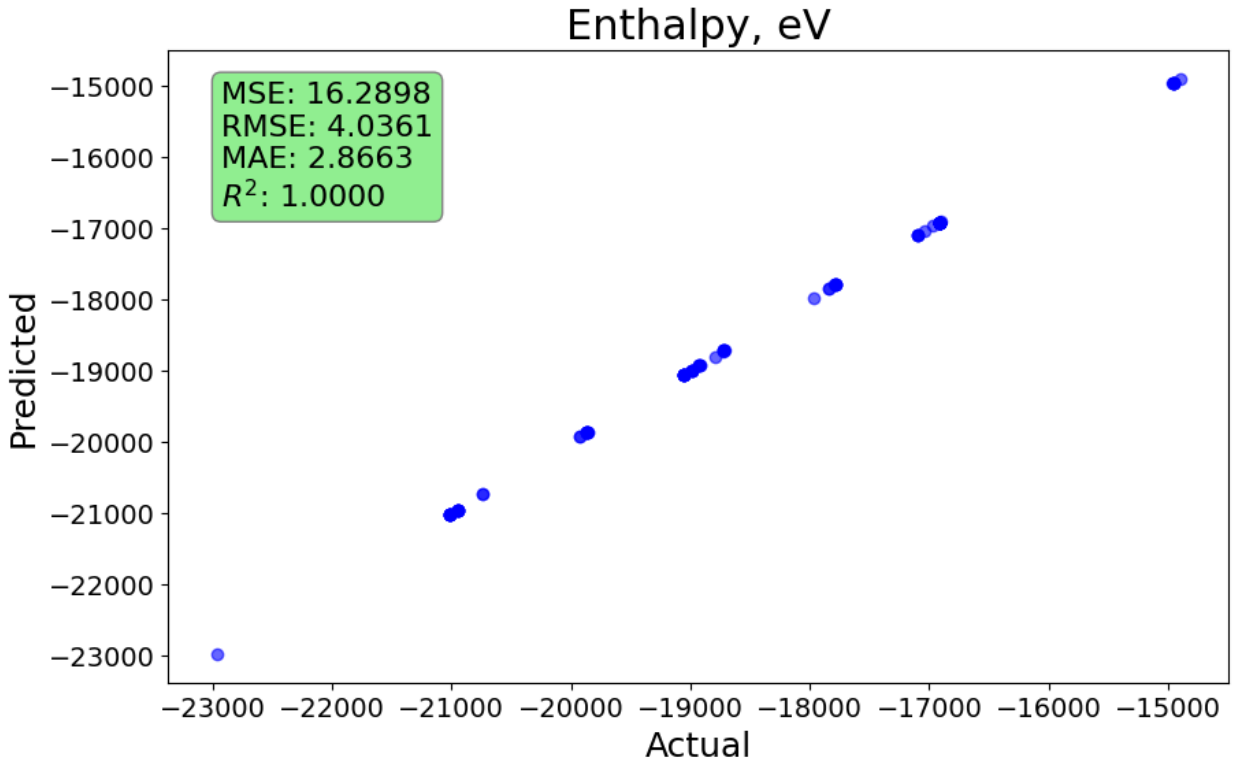
• **"metrics_text"** is a formatted string that displays the model evaluation metrics (MSE, RMSE, MAE, R^2), each with four decimal places

• **Lines 14 & 16** add this text to the upper left corner of the plot (**"transform=plt.gca().transAxes"** anchors it relative to the plot area, and to align the text at the top the **"verticalalignment='top'"** is specified). **"bbox"** – sets light green box surrounding the text for readability (with a gray border).

4. Save results to a csv

• **"results"** contains the DataFrame of Actual and Predicted values, which is subsequently saved as **"SLP(Act_vs_Pred).csv"** file.

The plot for Single Layer Perceptron with the accuracy metrics looks as follows:



3.3 Machine Learning: Results

After training and testing all the models, we can compare the accuracy metrics to determine the best-fitting model for our dataset (Table 2).

Table 2

Accuracy metrics across all models

ML model	MSE	RMSE	MAE	R^2
Linear Regression	2.7446	1.6567	1.4099	1.0000
Decision Tree	56.9403	7.5459	1.1456	1.0000
Single Layer Perceptron	16.2898	4.0361	2.8663	1.0000

• Mean Squared Error (MSE): The Linear Regression model has the lowest MSE at 2.7446, indicating it produces the least squared error on average. In contrast, the Decision Tree has a significantly higher MSE of 56.9403, suggesting it is less effective at predicting the target variable. The Single Layer Perceptron has a moderate MSE of 16.2898.

• Root Mean Squared Error (RMSE): RMSE values further reinforce the above, with Linear Regression having an RMSE of 1.6567, indicating smaller errors compared to the Decision Tree (7.5459) and the Single Layer Perceptron (4.0361). RMSE is sensitive to larger errors and emphasizes larger discrepancies in predictions.

• Mean Absolute Error (MAE): For MAE, the Decision Tree exhibits the lowest value (1.1456), indicating it may have fewer overall errors despite its higher MSE and RMSE. The Single Layer Perceptron has a higher MAE (2.8663) compared to Linear Regression (1.4099), suggesting a lower accuracy of its predictions.

• R-squared (R^2): All models have an R^2 value of 1.0000, indicating a perfect fit of the models to the training data. However, this metric alone can be misleading, particularly in cases of overfitting, as seen with the Decision Tree.

In summary, while all models show a perfect fit with an R^2 of 1.0000, the Linear Regression model is overall the most reliable based on MSE, RMSE, and MAE, suggesting it provides the best balance between the accuracy and the predictive performance. The Decision Tree, despite its lower MAE, performs poorly in terms of MSE and RMSE, indicating a potential overfitting. The Single Layer Perceptron, while not as effective as Linear Regression, performs better than the Decision Tree in terms of MSE and RMSE, but has a higher MAE. Therefore, for this particular dataset, Linear Regression would be the preferred model for making predictions.

REFERENCES

1 Borges R. M. et al. Quantum chemistry calculations for metabolomics: Focus review // Chemical reviews. – 2021. – Vol. 121. – №. 10. – P. 5633-5670. https://doi.org/10.1021/acs.chemrev.0c00901

2 Spiegel M., Gamian A., Sroka Z. A statistically supported antioxidant activity DFT benchmark—the effects of Hartree–Fock exchange and basis set selection on accuracy and resources uptake // Molecules. – 2021. – Vol. 26. – №. 16. – P. 5058. https://doi.org/10.3390/molecules26165058

3 Kar R. et al. Speeding-up Hybrid Functional-Based Ab Initio Molecular Dynamics Using Multiple Time-stepping and Resonance-Free Thermostat // Journal of Chemical Theory and Computation. – 2023. – Vol. 19. – №. 22. – P. 8351-8364. https://pubs.acs.org/doi/10.1021/acs.jctc.3c00964

4 Novikov I. S. et al. The MLIP package: moment tensor potentials with MPI and active learning // Machine Learning: Science and Technology. – 2020. – Vol. 2. – №. 2. – P. 025002. https://doi.org/10.1088/2632-2153/abc9fe

5 Pauletti M., Rybkin V. V., Iannuzzi M. Subsystem density functional theory augmented by a delta learning approach to achieve Kohn–Sham accuracy // Journal of Chemical Theory and Computation. – 2021. – Vol. 17. – №. 10. – P. 6423-6431. https://pubs.acs.org/doi/10.1021/acs.jctc.1c00592

6 Jumper J. et al. Highly accurate protein structure prediction with AlphaFold // nature. – 2021. – Vol. 596. – №. 7873. – P. 583-589. https://doi.org/10.1038/s41586-021-03819-2

7 Chemistry Prize [Electronic resource]. URL: https://www.nobelprize.org/prizes/chemistry/#:~:text=The%202024%20chemistry%20laureates,%E2%80%9Cfor%20protein%20structure%20prediction%E2%80%9D

8 Maulud D., Abdulazeez A. M. A review on linear regression comprehensive in machine learning // Journal of Applied Science and Technology Trends. – 2020. – Vol. 1. – №. 2. – P. 140-147. https://doi.org/10.38094/jastt1457

9 Dehghani A. A. et al. Decision tree algorithms // Handbook of hydroinformatics. – Elsevier, 2023. – P. 171-187. https://doi.org/10.1016/B978-0-12-821285-1.00004-X

10 Du K. L. et al. Perceptron: Learning, generalization, model selection, fault tolerance, and role in the deep learning era // Mathematics. – 2022. – Vol. 10. – №. 24. – P. 4730. https://doi.org/10.3390/math10244730

11 Welcome to Python.org [Electronic resource]. URL: https://www.python.org/

12 Pedregosa F. et al. Scikit-learn: Machine learning in Python // the Journal of machine Learning research. – 2011. – Vol. 12. – P. 2825-2830.

13 scikit-learn: machine learning in Python — scikit-learn 1.5.2 documentation [Electronic resource]. URL: https://scikit-learn.org/stable/

14 Abadi M. et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems // arXiv preprint arXiv:1603.04467. – 2016. https://doi.org/10.48550/arXiv.1603.04467

15 TensorFlow. TensorFlow [Electronic resource]. URL: https://www.tensorflow.org/

16 Ruddigkeit L. et al. Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17 // Journal of chemical information and modeling. – 2012. – Vol. 52. – №. 11. – P. 2864-2875. https://pubs.acs.org/doi/10.1021/ci300415d

17 Ramakrishnan R. et al. Quantum chemistry structures and properties of 134 kilo molecules // Scientific data. – 2014. – Vol. 1. – №. 1. – P. 1-7. https://doi.org/10.1038/sdata.2014.22

18 Dhinakaran V. Exploratory Data Analysis (EDA) and Data Visualization with Python. – 2018. http://dx.doi.org/10.17148/IJIREEICE.2024.12608

19 Galatro D., Dawe S. Exploratory Data Analysis // Data Analytics for Process Engineers: Prediction, Control and Optimization. – Cham : Springer Nature Switzerland, 2023. – P. 13-57. https://doi.org/10.1007/978-3-031-46866-7_2

20 Sandfeld S. Exploratory Data Analysis // Materials Data Science: Introduction to Data Mining, Machine Learning, and Data-Driven Predictions for Materials Science and Engineering. – Cham : Springer International Publishing, 2023. – P. 179-206. https://doi.org/10.1007/978-3-031-46565-9_9

21 Kumar A., Saharia M. Exploratory Analysis of Hydrological Data // Python for Water and Environment. – Singapore : Springer Nature Singapore, 2024. – P. 23-41. https://doi.org/10.1007/978-981-99-9408-3_4

22 Sahoo K. et al. Exploratory data analysis using Python // International Journal of Innovative Technology and Exploring Engineering. – 2019. – Vol. 8. – №. 12. – P. 4727-4735. https://doi.org/10.35940/ijitee.L3591.1081219

23 Quinn G. P., Keough M. J. Experimental design and data analysis for biologists. 2nd ed. – Cambridge university press, 2023. https://doi.org/10.1017/9781139568173.006

24 Starmans M. P. A. et al. Radiomics: data mining using quantitative medical image features // Handbook of medical image computing and computer assisted intervention. – Academic Press, 2020. – P. 429-456. https://doi.org/10.1016/B978-0-12-816176-0.00023-5

25 Nuzzo R. L. Histograms: A useful data analysis visualization // PM&R. – 2019. – Vol. 11. – №. 3. – P. 309-312. https://doi.org/10.1002/pmrj.12145

26 Vignesh V. et al. Data analysis using box and whisker plot for stationary shop analysis // 2017 International Conference on Trends in Electronics and Informatics (ICEI). – IEEE, 2017. – P. 1072-1076. https://doi.org/10.1109/ICOEI.2017.8300874

27 Cox N. J. Speaking Stata: Creating and varying box plots // The Stata Journal. – 2009. – Vol. 9. – №. 3. – P. 478-496. https://doi.org/10.1177/1536867X0900900309

28 Hirschauer N., Grüner S., Mußhoff O. The p-Value and Statistical Significance Testing // Fundamentals of Statistical Inference: What is the Meaning of Random Error?. – Cham : Springer International Publishing, 2022. – P. 63-96. https://doi.org/10.1007/978-3-030-99091-6_6

29 Di Leo G., Sardanelli F. Statistical significance: p value, 0.05 threshold, and applications to radiomics—reasons for a conservative approach // European radiology experimental. – 2020. – Vol. 4. – P. 1-8. https://doi.org/10.1186/s41747-020-0145-y

30 Nichelatti M. et al. L'intervallo di confidenza // Giornale di Clinica Nefrologica e Dialisi. – 2013. – Vol. 25. – №. 4. – P. 335-338. https://doi.org/10.33393/gcnd.2013.1070

31 Izonin I. et al. Towards data normalization task for the efficient mining of medical data // 2022 12th International Conference on Advanced Computer Information Technologies (ACIT). – IEEE, 2022. – P. 480-484. https://doi.org/10.1109/ACIT54803.2022.9913112

32 Aldi F. et al. Standardscaler's Potential in Enhancing Breast Cancer Accuracy Using Machine Learning // Journal of Applied Engineering and Technological Science (JAETS). – 2023. – Vol. 5. – №. 1. – P. 401-413. https://doi.org/10.37385/jaets.v5i1.3080

33 Srinivasan A. R. et al. Beyond RMSE: Do machine-learned models of road user interaction produce human-like behavior? // IEEE Transactions on Intelligent Transportation Systems. – 2023. – Vol. 24. – №. 7. – P. 7166-7177. https://doi.org/10.1109/TITS.2023.3263358

34 Hodson T. O. Root-mean-square error (RMSE) or mean absolute error (MAE): When to use them or not // Geoscientific Model Development. - 2022. – Vol. 15. - №. 14. - P. 5481-5487. https://doi.org/10.5194/gmd-15-5481-2022

35 Lin D. et al. Machine learning-based error compensation for high precision laser arbitrary beam splitting // Optics and Lasers in Engineering. – 2023. – Vol. 160. – P. 107245. https://doi.org/10.1016/j.optlaseng.2022.107245

36 Robeson S. M., Willmott C. J. Decomposition of the mean absolute error (MAE) into systematic and unsystematic components // PloS one. – 2023. – Vol. 18. – №. 2. – P. e0279774. https://doi.org/10.1371/journal.pone.0279774

37 Chicco D., Warrens M. J., Jurman G. The coefficient of determination R-squared is more informative than SMAPE, MAE, MAPE, MSE and RMSE in regression analysis evaluation // Peerj computer science. – 2021. – Vol. 7. – P. e623. https://doi.org/10.7717/peerj-cs.623

38 Redell N. Shapley decomposition of R-squared in machine learning models // arXiv preprint arXiv:1908.09718. – 2019. https://doi.org/10.48550/arXiv.1908.09718

39 Gao J. R-Squared (R2)–How much variation is explained? // Research Methods in Medicine & Health Sciences. – 2024. – Vol. 5. – №. 4. – P. 104-109. https://doi.org/10.1177/26320843231186398

40 Hodson T. O., Over T. M., Foks S. S. Mean squared error, deconstructed // Journal of Advances in Modeling Earth Systems. – 2021. – Vol. 13. – №. 12. – P. e2021MS002681. https://doi.org/10.1029/2021MS002681

41 Arnold C. et al. The role of hyperparameters in machine learning models and how to tune them // Political Science Research and Methods. – 2024. – Vol. 12. – №. 4. – P. 841-848. https://doi.org/10.1017/psrm.2023.61

42 Yang L., Shami A. On hyperparameter optimization of machine learning algorithms: Theory and practice // Neurocomputing. – 2020. – Vol. 415. – P. 295-316. https://doi.org/10.1016/j.neucom.2020.07.061

43 Weerts H. J. P., Mueller A. C., Vanschoren J. Importance of tuning hyperparameters of machine learning algorithms // arXiv preprint arXiv:2007.07588. – 2020. https://doi.org/10.48550/arXiv.2007.07588

44 Almarzooq H., bin Waheed U. Automating hyperparameter optimization in geophysics with Optuna: A comparative study // Geophysical Prospecting. – 2024. https://doi.org/10.1111/1365-2478.13484

45 Akiba T. et al. Optuna: A next-generation hyperparameter optimization framework // Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining. – 2019. – P. 2623-2631. https://doi.org/10.1145/3292500.3330701

46 DecisionTreeRegressor [Electronic resource]. URL: https://scikit-learn.org/dev/modules/generated/sklearn.tree.DecisionTreeRegressor.html

47 Singh J., Banerjee R. A study on single and multi-layer perceptron neural network // 2019 3rd International Conference on Computing Methodologies and Communication (ICCMC). – IEEE, 2019. – P. 35-40. https://doi.org/10.1109/ICCMC.2019.8819775

48 Bashetty S. et al. Optimizers in Deep Learning: A Comparative Study and Analysis // Int. J. Res. Appl. Sci. Eng. Technol. – 2022. – Vol. 10. – №. 12. – P. 1032-1039. https://doi.org/10.22214/ijraset.2022.48050

49 Abdulkadirov R., Lyakhov P., Nagornov N. Survey of optimization algorithms in modern neural networks // Mathematics. – 2023. – Vol. 11. – №. 11. – P. 2466. https://doi.org/10.3390/math11112466

50 Janjua J. I. et al. Activation Function Conundrums in the Modern Machine Learning Paradigm // 2023 International Conference on Computer and Applications (ICCA). – IEEE, 2023. – P. 1-8. https://doi.org/10.1109/ICCA59364.2023.10401760

51 Pantalé O. Comparing activation functions in machine learning for finite element simulations in thermomechanical forming // Algorithms. – 2023. – Vol. 16. – №. 12. – P. 537. https://doi.org/10.3390/a16120537

52 Chamarty A. Fine-Tuning of Learning Rate for Improvement of Object Detection Accuracy // 2020 IEEE India Council International Subsections Conference (INDISCON). – IEEE, 2020. – P. 135-141. https://doi.org/10.1109/INDISCON50162.2020.00038

53 Ma C. et al. Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't // arXiv preprint arXiv:2009.10713. – 2020. https://doi.org/10.48550/arXiv.2009.10713

54 Tian Y., Zhang Y. A comprehensive survey on regularization strategies in machine learning // Information Fusion. – 2022. – Vol. 80. – P. 146-166. https://doi.org/10.1016/j.inffus.2021.11.005

55 Moradi R., Berangi R., Minaei B. A survey of regularization strategies for deep models // Artificial Intelligence Review. – 2020. – Vol. 53. – №. 6. – P. 3947-3986. https://doi.org/10.1007/s10462-019-09784-7

56 Mazilu S., Iria J. L1 vs. L2 regularization in text classification when learning from labeled features // 2011 10th international conference on machine learning and applications and workshops. – IEEE, 2011. – Vol. 1. – P. 166-171. https://doi.org/10.1109/ICMLA.2011.85

57 Vidaurre Henche D. Regularization for sparsity in statistical analysis and machine learning : dis. – Informatica, 2012. https://doi.org/10.20868/UPM.thesis.14780

58 Michelucci U. Regularization // Applied Deep Learning with TensorFlow 2: Learn to Implement Advanced Deep Learning Techniques with Python. – Berkeley, CA : Apress, 2022. – P. 111-144. https://doi.org/10.1007/978-1-4842-8020-1_4

59 Sumi C. et al. Considerations about L2-and L1-norm regularizations for ultrasound reverberation characteristics imaging and vectoral Doppler measurement // 2022 44th Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC). – IEEE, 2022. – P. 3882-3886. https://doi.org/10.1109/EMBC48229.2022.9870991

60 Khakhar A., Buckman J. Neural Regression For Scale-Varying Targets // arXiv preprint arXiv:2211.07447. – 2022. https://doi.org/10.48550/arXiv.2211.07447

61 Hu C., Shi W. Impact of scaled image on robustness of deep neural networks // arXiv preprint arXiv:2209.02132. – 2022. https://doi.org/10.48550/arXiv.2209.02132

62 Schaul T. et al. Return-based scaling: Yet another normalisation trick for deep rl // arXiv preprint arXiv:2105.05347. – 2021. https://doi.org/10.48550/arXiv.2105.05347

Норматов Саадиаллах

Нестеров Павел Вячеславович

Алиев Тимур Алекберович

Тимралиева Александра Акбулатовна

Новиков Александр Сергеевич

Скорб Екатерина Владимировна

Practice-Oriented Introduction to Machine Learning: Linear Regression, Decision Tree, and Single Layer Perceptron models

Учебно-методическое пособие

В авторской редакции

Редакционно-издательский отдел Университета ИТМО

Зав. РИО Н.Ф. Гусарова

Подписано к печати

Заказ №

Тираж

Отпечатано на ризографе

Редакционно-издательский отдел

Университета ИТМО

197101, Санкт-Петербург, Кронверкский пр., 49, литер А