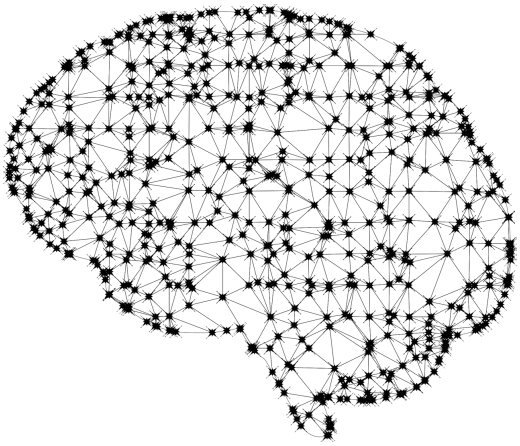
nbml

Computer Software for Data Analysis and Predictive Modelling with Artificial Intelligence Algorithms

Nikolaos P. Bakas[1], George Markou[2], Andreas Langousis[3], Spyros Lavdas[4], and Savvas Chatzichristofis[5]

[1]National Infrastructures for Research and Technology – GRNET. 7 Kifisias Avenue, 11523, Athens, Greece. nibas@grnet.gr
[2]Civil Engineering Department, University of Pretoria, Hatfield Campus, Pretoria, 0028. george.markou@up.ac.za
[3]Department of Civil Engineering, University of Patras, 26504, Patras, Greece.andlag@upatras.gr
[4]Department of Computer Science, Athens Metropolitan College, Marousi, Greece. slavdas@mitropolitiko.edu.gr
[5]Neapolis University Pafos. Intelligent Systems Laboratory, Department of Computer Science. Pafos, Cyprus. s.chatzichristofis@nup.ac.cy.

v1.00, 17 March 2023



Contents

1 Introduction

nbml is a generic-purpose software for the analysis of Tabular Datasets, and building Artificial Intelligence (AI) and Machine Learning (ML) models. It aims to support experts in the field and people from other disciplines who want to analyse their data. In this document, we explain the open software's main functionality, along with implementation examples. The document also comprises references with Basic and Advanced Literature for Data Analysis with AI algorithms, including links to the corresponding sources, where you can allocate the respective manuscript in which the dataset was first published. You may read the part of each referenced article as indicated, and also go through the optional literature that is also provided in this document.

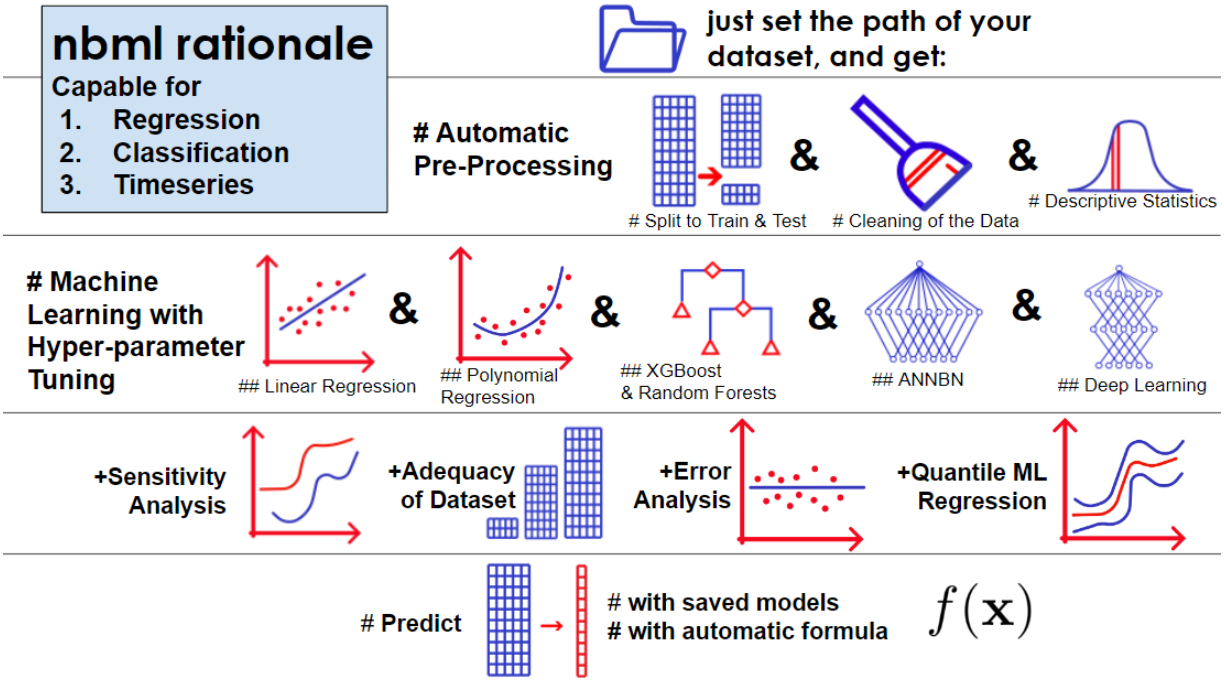


Figure 1: **nbml** features

1.1 Introductory Literature

The aim of AI algorithms is to construct a mathematical model that describes in an optimal manner the given input data with the corresponding output data that are found within a specific dataset. Afterward, we may utilize the model to interpret the associations among variables and/or predict the values of data that are out-of-sample. This means that the developed predictive model will be used to provide predictions on data that were not used to train or test the predictive model. For additional reading material, please turn to the following literature:

- Can we predict? Read the introduction section in [17].
- For forecasting and uncertainty, please read the abstract of [24].
- (Optional) AI algorithms lacking a solid theoretical background [47].
- (Optional) The theoretical foundation and analysis are often vague [46].
- A collection of Statistical and ML Concepts videos, click Statquest Videos.

1.2 Sources for Coding

Some insightful and helpful material related to source coding can be found through the following links:

- W3schools - Python ML Getting Started
- Generic PyTorch Code for Regression
- Generic PyTorch Code for Computer Vision
- Machine Learning algorithms for structured datasets with the Julia Language

A good start for coding ML algorithms, is to read and implement the following from https://www.w3schools.com/python/python_ml_getting_started.asp:

1. Machine Learning
2. Getting Started
3. Mean Median Mode
4. Standard Deviation
5. Percentile
6. Data Distribution
7. Normal Data Distribution
8. Scatter Plot
9. Linear Regression
10. Multiple Regression
11. Scale
12. Train/Test
13. Logistic Regression
14. Categorical Data
15. Cross Validation

1.3 Applications

Several applications and different types of problems have been investigated herein, where AI and ML algorithms were used to develop predictive models that would allow us to investigate further the input-target relationships. Some of those applications can be found in the following references:

- Read Sections 3, 5, and 6 from [22], for Artificial Neural Networks and Regression for Structural Design against Torsion.
- Read Accuracy Metrics from [20], Section 5, Models' description and diagnostics.
- (Optional) Artificial Neural Networks for the evaluation of Ratio of Torsion [23]
- (Optional) Multinomial logistic regression model, for risk ratios [21]
- (Optional) Analysis of Variance (ANOVA) and confidence intervals [19].
- (Optional) Clustering of variables [25].
- (Optional) Read abstract from [27] EcoCement: A Novel Composite Material For The Construction Industry. Identification Of An Optimal Recipe Using Neural Networks.
- (Optional) Read abstract from [28] Regression analysis vs genetic algorithms: computational efficiency assessment on the design of proindustry project SSDC isolators under Incremental dynamic loading

1.4 Computational Analysis of Literature

We strongly recommend systematically analysing the literature on your topic of study, before you start performing research on a specific scientific field. This regards the stage after the identification of your thematic area, as well as the evidence-based extraction of useful insights. Given that the bibliography is sometimes massive in specific scientific areas, computational analysis, AI and ML algorithms can also be used to assist with this tedious task.

Different research work can be found in the following citations that relate to the bibliometric literature review of adaptive learning systems [2, 3, 15, 16, 18, 26].

1.5 Datasets

1.5.1 Regression

The research work that was performed within the Civil Engineering discipline has allowed us to develop different datasets that can be used to develop several predictive models. These datasets referred to problems such as calculating the shear strength of slender and deep beams without stirrups, and the computation of the fundamental period of steel and reinforced concrete (RC) structures. You may use the following datasets, located in the folder "datasets", where each reference is also provided where the dataset was first published, as described below:

1. Deep Beams with FRP 2022.xlsx [4] Number of Rows: 84, Input Columns: B-I, Target Column: J
2. Fundamental RC Structures 2022.xlsx [5]: Number of Rows: 790, Input Columns: A-F, Target Column: G
3. Curved Beam 2022.xlsx [6]: Number of Rows: 1 320, Input Columns: A-J, Target Column: K
4. Fundamental Steel 2022.xlsx [7]: Number of Rows: 1 152, Input Columns: A-F, Target Column: G
5. RC Slender beams Raw dataset.xlsx [8]: Number of Rows: 35 849, Input Columns: A-J, Target Column: K
6. Fundamental TrainTest and Validation data raw 2021.xlsx [11]: for Train/Test $->$ Number of Rows: 475, Input Columns: A-F, Target Column: G, and for Validation $->$ Number of Rows: 60, Input Columns: A-F, Target Column: G
7. Dataset 98308 Steel Frames with SSI.xlsx [Under Review]: Number of Rows: 98 308, Input Columns: A-F, Target Column: G

1.5.2 Classification

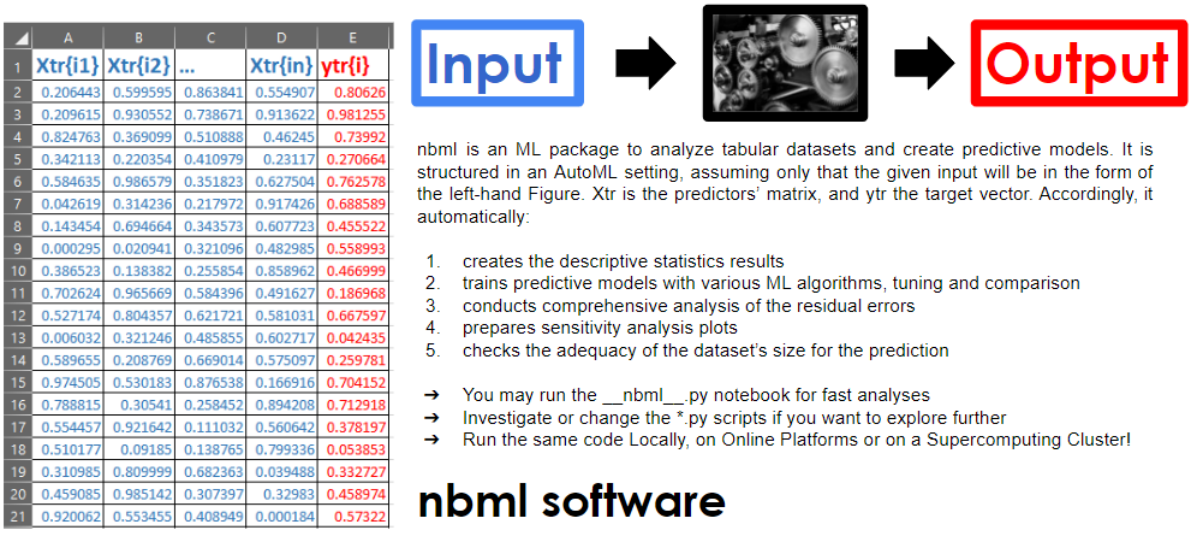
You may also test the software, by using
https://www.kaggle.com/datasets/datazng/telecom-company-churn-rate-call-center-data
for predicting the Churn Rate in a Telecom Company. Put the Telecom Churn Rate Dataset.xlsx in the ROOT_DIR, delete the first column, and you may directly run nbml.

2 Main User Environment

2.1 Program's Rationale Remarks

The Software is designed to run the main __nbml__.py script, which calls the associated *.py files. The rationale is formatted in an input-output setting as shown in Figure 2.

By running each one of the ### **blocks** sequentially, you can obtain the entire analysis and predictive models. All the results are saved into the folders, under the directory where the Project's .xlsx file is located. If one would like to delete all the results and run the analysis again from scratch, one can simply delete all the folders within the workspace and run the code again. The software will automatically re-create the results, given that the folder is now empty and has no files. Each analysis prints the results that are also saved in **output.html**.



	A	B	C	D	E
1	Xtr{i1}	Xtr{i2}	...	Xtr{in}	ytr{i}
2	0.206443	0.599595	0.863841	0.554907	0.80626
3	0.209615	0.930552	0.738671	0.913622	0.981255
4	0.824763	0.369099	0.510888	0.46245	0.73992
5	0.342113	0.220354	0.410979	0.23117	0.270664
6	0.584635	0.986579	0.351823	0.627504	0.762578
7	0.042619	0.314236	0.217972	0.917426	0.688589
8	0.143454	0.694664	0.343573	0.607723	0.455522
9	0.000295	0.020941	0.321096	0.482985	0.558993
10	0.386523	0.138382	0.255854	0.858962	0.466999
11	0.702624	0.965669	0.584396	0.491627	0.186968
12	0.527174	0.804357	0.621721	0.581031	0.667597
13	0.006032	0.321246	0.485855	0.602717	0.042435
14	0.589655	0.208769	0.669014	0.575097	0.259781
15	0.974505	0.530183	0.876538	0.166916	0.704152
16	0.788815	0.30541	0.258452	0.894208	0.712918
17	0.554457	0.921642	0.111032	0.560642	0.378197
18	0.510177	0.09185	0.138765	0.799336	0.053853
19	0.310985	0.809999	0.682363	0.039488	0.332727
20	0.459085	0.985142	0.307397	0.32983	0.458974
21	0.920062	0.553455	0.408949	0.000184	0.57322

Figure 2: **nbml** features

2.2 How to Install

1. In order to install and run the software, you will need to have Visual Studio Code for Windows 64bit installed on your computer. Download and Install Visual Studio Code: https://code.visualstudio.com/
2. Install Python from here https://www.python.org/downloads/. Please select a version compatible with PyTorch (e.g. Python 3.7-3.9) as described here: https://pytorch.org/get-started/locally/
3. In Visual Studio Code, go to Extensions (left-hand top) and search for Python. Install there the Python Extension.
4. Install Libraries
   - Open a Command Prompt (cmd) as administrator.
   - Change directory to "Scripts" with the command:
     cd C:\Users\YOURUSERNAME\AppData\Local\Programs\Python\PythonXX\Scripts
   - Place the file "requirements.txt" into the above directory.
   - Return to cmd and, type and execute the command **pip install -r requirements.txt**
   - Install PyTorch for Windows, where the command you should type in cmd is
     pip3 install torch torchvision torchaudio
     Find the correct command for your computer by reading the instruction found at https://pytorch.org/

Alternatively, you may install the libraries within vscode, by starting Python binary, by typing **Shift+Enter** on Windows, or running in the terminal **python3** on Mac, and then execute the following commands:

```
import pip
pip.main(['install', 'numpy'])
pip.main(['install', 'adjustText'])
pip.main(['install', 'nbconvert'])
pip.main(['install', 'pandas'])
pip.main(['install', 'seaborn'])
pip.main(['install', 'scipy'])
pip.main(['install', 'scikit-learn'])
pip.main(['install', 'statsmodels'])
```

```
pip.main(['install', 'xgboost'])
pip.main(['install', 'openpyxl'])

import subprocess
subprocess.call(['pip3', 'install', 'torch'])
subprocess.call(['pip3', 'install', 'torchvision'])
subprocess.call(['pip3', 'install', 'torchaudio'])
```

Everything should be ready. If you used another version of Python in VSCode, change it with Ctrl+Shif+P − > Python: Select Interpreter and select the python.exe in
cd C:\Users\YOURUSERNAME\AppData\Local\Programs\Python\PythonXX
It is very important to install the dependencies that are required by to code to execute the software on your computer successfully. For this reason, the following libraries should be installed (using the above-mentioned commands), prior to the execution of the code:

1. pip install adjustText [37]
2. pip install nbconvert [38]
3. pip install pandas [39]
4. pip install seaborn [36]
5. pip install matplotlib [35]
6. pip install scipy [34]
7. pip install scikit-learn [32, 33]
8. pip install statsmodels [31]
9. pip install xgboost [30]
10. pip install openpyxl [40]
11. pip3 install torch torchvision torchaudio [41]

The instructions thus far were referring to the Windows operating system and Visual Studio Code (vs-code). You may use another Operating System, such as Mac or Linux, as well as Source Code Editor following similar steps.

2.3 How to Use the Software's Python Code

In order to be able to execute the software's code, which is written in Python language, you need to:

1. Put all *.py files in any directory.
2. Under "ROOT_DIR" there should be only one working .xlsx dataset file.
3. If the objective is to predict for new out-of-sample data, place the corresponding excel[1] file in the "Predict" folder under ROOT_DIR, and run the "Open Dataset" part of the code first.

[1] https://www.microsoft.com/fi-fi/microsoft-365/excel

2.4 To run the Software's Code on a Cluster

So as to execute the software on a cluster, one has to follow the steps below:
allocate a node e.g.

salloc –nodes=1 –ntasks=1 –ntasks-per-node=1 –cpus-per-task=128
–time=00:59:00 –partition=cpu –account=${account_name} –qos=dev

salloc –nodes=1 –ntasks=4 –ntasks-per-node=4 –gpus-per-task=1
–time=01:59:00 –partition=gpu –account=${account_name} –qos=dev

Then you need to do the following
1) start with shift+enter a new python
2) a new srun will be activated, with python running on the frontend 3) ctrl+z to stop python
4) ssh to the allocated node
load modules (e.g. for MeluXina cluster https://docs.lxp.lu/)
module load env/staging/2022.1
module load Python/3.10.4-GCCcore-11.3.0
ml SciPy-bundle/2022.05-foss-2022a
ml PyTorch/1.12.0-foss-2022a-CUDA-11.7.0
ml IPython/8.5.0-GCCcore-11.3.0
python

Use the following to ensure the correct node
shift+enter in the script
import sys
print(sys.executable)
import socket
hostname = socket.gethostname()
hostname

Use the following to view GPU usage. You can use the initial srun (right hand, bottom side in VSCode[2])

watch -n1 'nvidia-smi –query-gpu=index,name,utilization.gpu,memory.used,memory.total –format=csv'

put this before pytorch train epochs loop

with torch.backends.mkl.verbose(torch.backends.mkl.VERBOSE_OFF):

You may also use slurm on CPU or GPU by using:

Listing 1: slurm on CPU

```
#!/bin/bash -l
#SBATCH --nodes=1                    # number of nodes
#SBATCH --ntasks=1                   # number of tasks
#SBATCH --ntasks-per-node=1          # number of tasks per node
```

[2] https://code.visualstudio.com/

```
#SBATCH --cpus-per-task=128                # number of cores per task
#SBATCH --time=00:30:00                    # time (HH:MM:SS)
#SBATCH --partition=cpu                    # partition
#SBATCH --account=${account_name}          # project account
#SBATCH --qos=dev


module load env/staging/2022.1
module load Python/3.10.4-GCCcore-11.3.0
ml SciPy-bundle/2022.05-foss-2022a
ml PyTorch/1.12.0-foss-2022a-CUDA-11.7.0
ml IPython/8.5.0-GCCcore-11.3.0

cd nbml
python __nbml__.py
```

Listing 2: slurm on GPU

```
#!/bin/bash -l
#SBATCH --nodes=1                          # number of nodes
#SBATCH --ntasks=4                         # number of tasks
#SBATCH --ntasks-per-node=4                # number of tasks per node
#SBATCH --gpus-per-task=1                  # number of cores per task
#SBATCH --time=00:30:00                    # time (HH:MM:SS)
#SBATCH --partition=gpu                    # partition
#SBATCH --account=${account_name}          # project account
#SBATCH --qos=dev


module load env/staging/2022.1
module load Python/3.10.4-GCCcore-11.3.0
ml SciPy-bundle/2022.05-foss-2022a
ml PyTorch/1.12.0-foss-2022a-CUDA-11.7.0
ml IPython/8.5.0-GCCcore-11.3.0

cd nbml
python __nbml__.py
```

2.5 To Run the Code on Google Colab

The developed open-source code of the software provided through the .py files has the ability to be executed directly on Google Colab. This can be performed by following the steps that are described in this section:

1. Upload the code on https://colab.research.google.com/
2. Split the code into cells (optional)
3. Upload all script files
4. Upload your dataset
5. Run the code!

3 Main Flow of the Software's Code: __nbml__.py

When all required packages have been installed, you may execute the code found in the main __nbml__.py file, which sequentially calls the functions defined in the next sections. You may run the ### blocks sequentially, using **Shift+Enter** in VSCode. In Section 11 a numerical implementation is presented, where the main functionalities of the software are presented.

The only this you need to define is the path where your excel file is, in **ROOT_DIR**.

```
#
    ##############################################################################################

# More instructions in README.md and __nbml__.pdf
#
    ##############################################################################################
```

```
#
    ##############################################################################################

# Define the problem
# Simply define the main parameters here. The code will automatically produce
    the corresponding graphs and tables.
ROOT_DIR = ""
# A *****directory without the filename***** with only one excel file.
# The *.xlsx file shhould comprise with all the independent variables at the
    first $n$ columns, followed by the target variable as the last column.
# For Windows, please use *****\\***** separators and remeber to *****add the
    \\ at the end*****.
# For Linux please use /.../.../ format
LOGISTIC_REGR = False # If True do classification
PERMUTE_TRAIN_TEST = True # If True split the data into training/testing sets
    randomly, after shuffling.
# If False, Top rows are train and bottom test, which is helpfull for time
    series data.
#
    ##############################################################################################
```

```
#
    ##############################################################################################

# import Libraries
from import_libraries import *
import import_libraries, misc_functions, descriptive_statistics,
    ml_linear_regression, ml_nlregr, ml_xgboost
import ml_ANNBN, ml_random_forests, ml_DANN
reload(import_libraries); reload(misc_functions); reload(
    descriptive_statistics)
```

```
reload(ml_linear_regression); reload(ml_nlregr); reload(ml_xgboost); reload(
    ml_ANNBN); reload(ml_random_forests); reload(ml_DANN)
#
    ##########################################################################################



#
    ##########################################################################################

# Open the Dataset and Split to Train & Test
test_ratio = 0.3 #The ratio of the data to be used for testing (0-1). Usually
    0.2-0.3
random_seed = 0 #The random seed to be used for the random number generator.
Xtr, Xte, ytr, yte, features_names, target_name = misc_functions.
    split_train_test(PERMUTE_TRAIN_TEST, test_ratio, random_seed, ROOT_DIR)
misc_functions.create_all_directories(ROOT_DIR, PERMUTE_TRAIN_TEST)
#
    ##########################################################################################



#
    ##########################################################################################

# Clean the Data
Xte,yte = misc_functions.delete_missing_values_rows(Xte,yte)
Xtr,ytr = misc_functions.delete_missing_values_rows(Xtr,ytr)
# Xtr,ytr = misc_functions.delete_identical_rows(Xtr,ytr)
Xtr, Xte, features_names = misc_functions.check_fix_multicolinearity(Xtr, Xte,
     features_names, ROOT_DIR)
# Xtr, ytr, features_names = misc_functions.make_lags(Xtr, ytr, features_names
    , target_name, lags=51)
# Xte, yte, features_names = misc_functions.make_lags(Xte, yte, features_names
    , target_name, lags=51)
#
    ##########################################################################################



#
    ##########################################################################################

# Descriptive Statistics
descriptive_statistics.descriptive_statistics(Xtr, Xte, features_names,
    ROOT_DIR)
descriptive_statistics.plot_short_tree(Xtr, ytr, features_names, target_name,
    ROOT_DIR)
descriptive_statistics.plot_pdf_cdf_all(Xtr, ytr, features_names, target_name,
     ROOT_DIR)
descriptive_statistics.plot_all_by_all_correlation_matrix(Xtr, ytr,
```

```
    features_names, target_name, ROOT_DIR)
# descriptive_statistics.export_descriptive_per_bin(Xtr,Xte,ytr,yte,
    features_names, target_name, ROOT_DIR)
# descriptive_statistics.plot_all_timeseries(Xtr, features_names, ytr,
    target_name, "Train", ROOT_DIR)
# descriptive_statistics.plot_all_timeseries(Xte, features_names, yte,
    target_name, "Test", ROOT_DIR)
#
    ##########################################################################################



#
    ##########################################################################################

# Machine Learning
#
    ##########################################################################################



## Linear Regression
ml_linear_regression.do_regression(Xtr, Xte, ytr, yte, features_names,
    target_name, ROOT_DIR, LOGISTIC_REGR)


## Polynomial Regression
ml_nlregr.do_nlregr(Xtr, Xte, ytr, yte, features_names, target_name,
    LOGISTIC_REGR, PERMUTE_TRAIN_TEST, ROOT_DIR)


## XGBoost
__thres_early_stop__ = 1e-2
__thres_min_tune_rounds__ = 100
ml_xgboost.do_xgboost(Xtr,Xte,ytr,yte,features_names,target_name,
    __thres_early_stop__,__thres_min_tune_rounds__,PERMUTE_TRAIN_TEST,
    LOGISTIC_REGR,ROOT_DIR)
ml_xgboost.do_QuantileGradientBoostingRegressor(ROOT_DIR, Xtr, Xte, ytr, yte,
    target_name)


## Random Forests
__thres_early_stop__ = 1e-2
__thres_min_tune_rounds__ = 100
ml_random_forests.do_random_forests(Xtr,Xte,ytr,yte,features_names,target_name
    ,__thres_early_stop__,__thres_min_tune_rounds__,PERMUTE_TRAIN_TEST,
    LOGISTIC_REGR,ROOT_DIR)


## ANNBN
ml_ANNBN.do_ANNBN(Xtr, Xte, ytr, yte, features_names, target_name,
    PERMUTE_TRAIN_TEST, LOGISTIC_REGR, ROOT_DIR, min_obs_over_neurons=20)
```

```
## Deep Learning
__thres_early_stop__ = 1e-2
__thres_min_tune_rounds__ = 100
ml_DANN.do_DANN(Xtr, ytr, Xte, yte, features_names, target_name,
    __thres_early_stop__, __thres_min_tune_rounds__, PERMUTE_TRAIN_TEST,
    LOGISTIC_REGR, ROOT_DIR)


#
    ##########################################################################################

# Sensitivity Analysis
misc_functions.gather_all_sensitivity_curves(Xtr, features_names, target_name,
     ROOT_DIR)
#
    ##########################################################################################



#
    ##########################################################################################

# Predict: Please run the first #####2 blocs##### at the top of this file,
    these are:
# ####Define the problem#####
# ####import Libraries#####
# The *.xlsx file to be used for prediction, should be in the #####"Predict
    "##### folder, inside #####ROOT_DIR##### directory.
#
    ##########################################################################################

Xout, yout, features_names, target_name = misc_functions.
    read_out_of_sample_data(ROOT_DIR)
Xout, yout = misc_functions.delete_missing_values_rows(Xout, yout)
#
    ##########################################################################################

ml_linear_regression.predict_linregr(Xout, yout, target_name, LOGISTIC_REGR,
    ROOT_DIR)
#
    ##########################################################################################

ml_xgboost.predict_xgboost(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR)
ml_xgboost.predict_quantile_gb(ROOT_DIR, Xout)
#
    ##########################################################################################

ml_random_forests.predict_rf(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR)
#
    ##########################################################################################

ml_ANNBN.predict_ANNBN(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR)
#
```

```
##########################################################################################

ml_DANN.predict_DANN(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR)
#
##########################################################################################
```

4 Structure and Use of a DataSet for Training and Testing

4.1 Input .xlsx Dataset Files

Each file, along with the corresponding results and models, is considered and handled as an independent project. Therefore, the dataset should be found within an .xlsx file that will consist of all the independent variables at the first set of columns n, followed by the target variable positioned at the last column. At the top of each column, the first cell should contain the corresponding parameter name. For example, Figure 3 shows the general format that any dataset should have in order to be able to be used for training testing and prediction.

This is the only input required, and the following analyses are all performed automatically based on the standard parameters that you may change as per the next sub-section. Using short names for the variables is a good practice, as they will be used in the later figures and formulae development. Also, using only letters, dashes and numbers is advisable when defining the parameters' names. Furthermore, please use a plain spreadsheet without any colour or border format, as shown in Figure 3. In Figure 4, the initial commands can be seen where the definition of the folder that contains the dataset takes place and the opening of the .xlsx file is performed.

	A	B	C	D	E
1	Xtr{i1}	Xtr{i2}	...	Xtr{in}	ytr{i}
2	0.206443	0.599595	0.863841	0.554907	0.80626
3	0.209615	0.930552	0.738671	0.913622	0.981255
4	0.824763	0.369099	0.510888	0.46245	0.73992
5	0.342113	0.220354	0.410979	0.23117	0.270664
6	0.584635	0.986579	0.351823	0.627504	0.762578
7	0.042619	0.314236	0.217972	0.917426	0.688589
8	0.143454	0.694664	0.343573	0.607723	0.455522
9	0.000295	0.020941	0.321096	0.482985	0.558993
10	0.386523	0.138382	0.255854	0.858962	0.466999
11	0.702624	0.965669	0.584396	0.491627	0.186968
12	0.527174	0.804357	0.621721	0.581031	0.667597
13	0.006032	0.321246	0.485855	0.602717	0.042435
14	0.589655	0.208769	0.669014	0.575097	0.259781
15	0.974505	0.530183	0.876538	0.166916	0.704152
16	0.788815	0.30541	0.258452	0.894208	0.712918
17	0.554457	0.921642	0.111032	0.560642	0.378197
18	0.510177	0.09185	0.138765	0.799336	0.053853
19	0.310985	0.809999	0.682363	0.039488	0.332727
20	0.459085	0.985142	0.307397	0.32983	0.458974
21	0.920062	0.553455	0.408949	0.000184	0.57322

Figure 3: nbml_AutoML Input Data Format

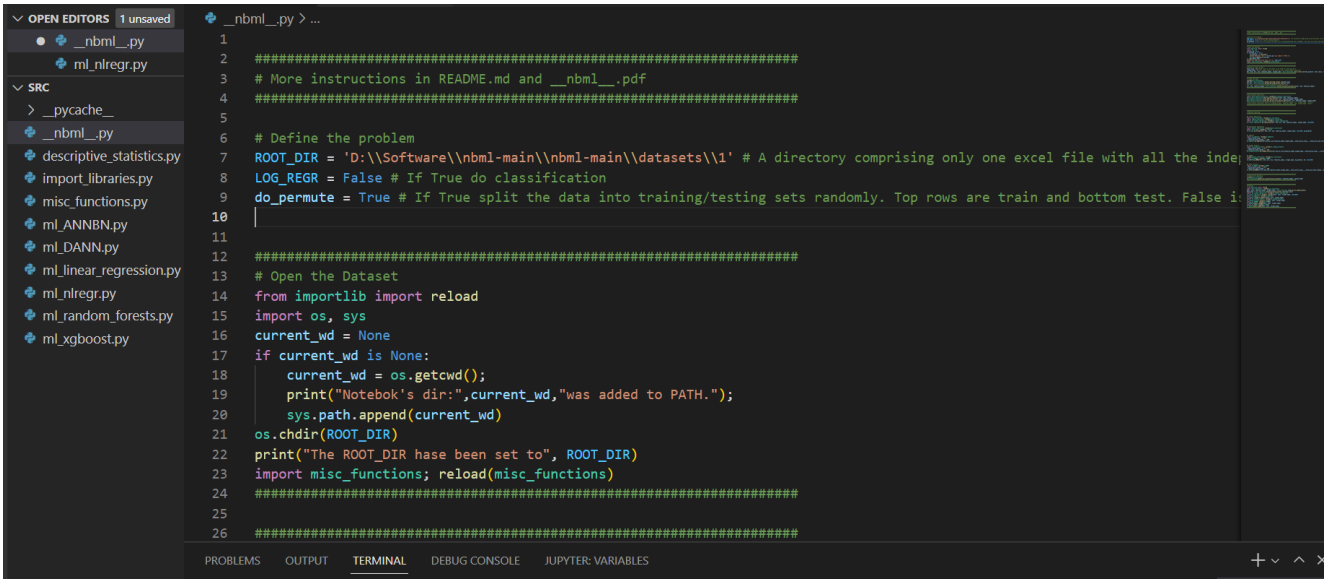


Figure 4: First Commands of the Code: Lines 1-23

4.2 Split the Dataset into Train-Test Sets

The part of the code shown in Figure 5 performs the split according to the ratio defined in the parameters. It will create two new datasets on the RAM of your PC and in this case, 80% of the data will be used for training and 20% will be used to test the developed predictive model.

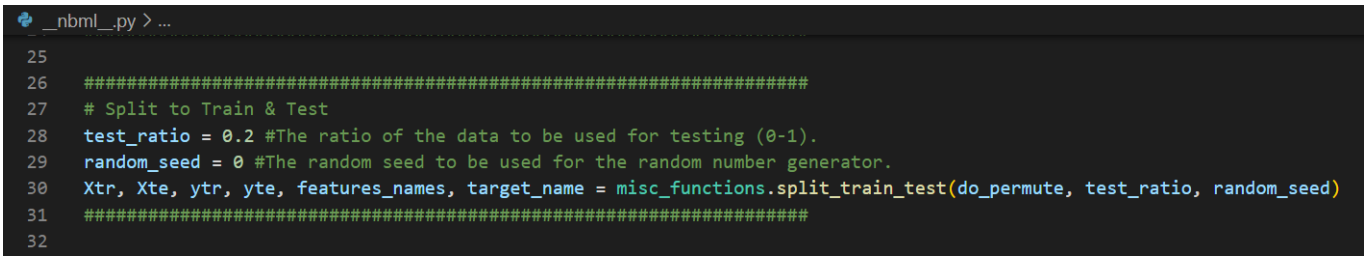


Figure 5: Split the Dataset into Training and Testing sets: Lines 27-30

4.3 Cleaning

The next step of the code foresees the cleaning of the Dataset as shown in Figure 6. In this section, we provide functions for the automatic cleaning of the dataset that can be activated and deactivated accordingly. For example, in the case of Figure 6, we assume that the rows within the dataset that have missing values will be deleted, while the check for rows within the dataset that are identical is not performed. The check for finding identical rows can be computationally demanding for large datasets, therefore, in the case that we are certain that the dataset does not have identical rows, this check can be avoided. Rows are found within the train and test sets, where we keep only the unique ones. Identical in terms of input parameters and output target are rows that are deleted according to this function.

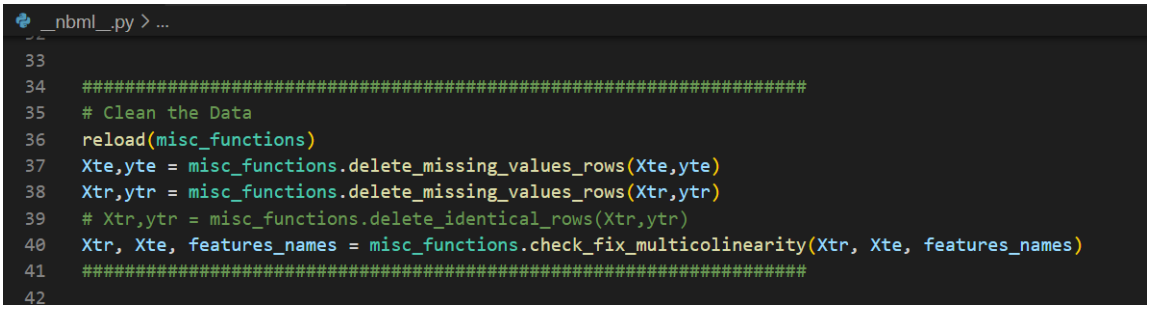


Figure 6: Dataset Cleaning: Lines 36-40

Furthermore, the multicolinearity check is performed according to the command in line 40 of the code. A check for colinear features is performed by:

1. computing the rank k, of the training Matrix,
2. applying QR-Factorization, and selecting the first pivoted k columns.
3. replacing Train and Test sets with the new ones with non-co-linear columns.

5 Descriptive Statistics

After the operations of reading the dataset, dividing it into training and testing sets, and cleaning the data provided by executing the functions that are shown in Figure 6, the code proceeds with the statistical description of the data. By running the section of the code shown in Figure 7, all descriptive statistics will be computed along with illustrative Figures, and saved in the corresponding folder.

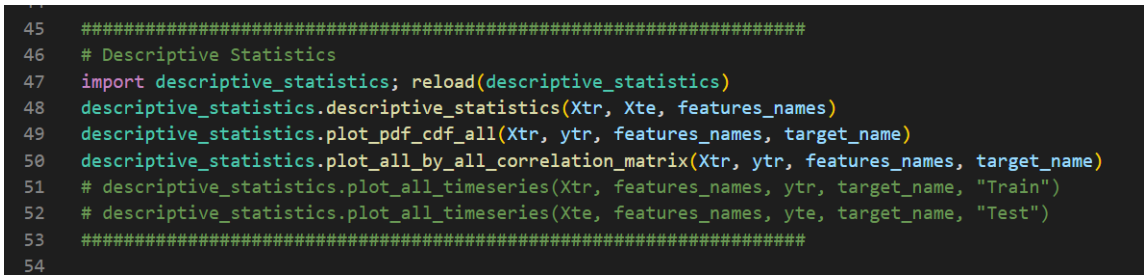


Figure 7: Descriptive Statistics Code: Lines 47-52

Particularly, the "Descriptive Statistics" part of the code computes the following:

1. Histograms of all features and target, with the corresponding Cumulative Distribution Functions.
2. Correlations (Pearson), of
   - Features Pair-wisely. For more information, you may refer to the diagrams in [44]. The mathematical part is optional.
   - Input features with output target.
   - Correlations map through the use of a heuristic algorithm defined in [9, 10]. This allows for the conversion of the correlation matrix to the corresponding map, such that the distances among the elements represent their correlations. Input features with close distance, tend to have higher correlations.
3. Timeseries's plots if the variable do_permute == False

The Descriptive Statistics regard mainly the train set. A table, comparing the corresponding values of the test set is also developed. For additional information, refer to the code in file **descriptive_statistics.py**, and Section 17 (Appendix: Code for descriptive statistics.py).

6 Machine Learning Models

It must be noted at this point that when we refer to "features" we refer to input parameters and for the case of the output target value at each row, we will refer to it as the "target". At this stage, all required preparation was executed, and the processes that relate to the preparation of the dataset at hand were successfully performed. Therefore, we are now ready to engage with the training and testing by using the dataset that we have defined at the beginning of the code.

In order to train, the "Machine Learning" part of the code has to be executed, where we use the following models for approximating features-target relationships (development of predictive models). For each predictive model, we compute the following:

1. The accuracy among prediction and target variable, for the train and test sets.
2. Error analysis for the train and test sets. Particularly, the
   - Residual Errors vs Target diagrams
   - Probability Density Functions, and Cumulative Density Functions,

   This way we may identify specific patterns occurring in the prediction, and, hence, the generalization capability and reliability of the model.

Furthermore, we compute the number of train values found in that bin vs the corresponding mean absolute error in test set per bin. Ultimately, we aggregate all error metrics for all methods in a unified table, for comparison reasons, and as it will be shown in the numerical implementation section.

Following, the different ML and AI algorithms that the user can implement to develop their predictive models are presented.

6.1 Linear Regression

In this section, we discuss the code related to the Linear Regression analysis. The linear regression model is considered a baseline for the next ML and AI models since this models is simplistic and always found to derive the highest error metrics. Typically linear regression models have lower accuracy, however, they are robust to noise. The part of the code that has to be executed is seen in Figure 8

```
61  ## Linear Regression
62  import misc_functions; reload(misc_functions)
63  import import_libraries; reload(import_libraries)
64  import ml_linear_regression; reload(ml_linear_regression)
65  ml_linear_regression.do_regression(Xtr, Xte, ytr, yte, features_names, target_name, LOG_REGR)
66
```

Figure 8: Linear Regression Code: Lines 62-65

6.1.1 Literature for Linear Regression Models

For the reader that would like to learn more information on this type of model they can refer to:

- Correlation Coefficient for two variables [44]. See the diagrams. The mathematical part is optional for beginners.
- Basic theory for Least Squares Fitting [43]. The linear regression models are considered as a baseline. Study the figures and text found within these references. The mathematical part is optional.
- Interesting Equations for Least Squares (clickable link)
- Assumptions of Linear Regression (clickable link)

6.1.2 Linear Regression Theory

This section will present the basic mathematical formulation of the linear regression theory. Let $\mathbf{ytr} = ytr_i$ a set of the training values of the target variable, and $\mathbf{Xtr} = \{Xtr_{i1}, Xtr_{i2}, \ldots, Xtr_{in}\}$ the matrix comprising n number of samples, for each predictor. $i \in \{1, 2, \ldots, m\}$ is the number of observations, and $j \in \{1, 2, \ldots, n\}$, the number of predictors.

The linear regression model assumes a predictors-target relationship in the form of

$$ytr_i = \alpha_0 + \alpha_1 Xtr_{i1} + \cdots + \alpha_n Xtr_{in} + \varepsilon_i = \mathbf{Xtr}_i^{\mathsf{T}} \times \alpha + \varepsilon_i. \tag{1}$$

Hence, we may write in matrix form the linear connection between the error, the predictors, the constants $\boldsymbol{\alpha}$ and the target as

$$\mathbf{ytr} = \mathbf{Xtr} \times \boldsymbol{\alpha} + \boldsymbol{\varepsilon}, \tag{2}$$

where $\boldsymbol{\varepsilon}$ is the vector comprising all the errors of the model,

$$\boldsymbol{\varepsilon} = \mathbf{ytr} - \hat{\mathbf{ytr}}. \tag{3}$$

Hence, we have that

$$\mathbf{ytr} = \begin{bmatrix} ytr_1 \\ ytr_2 \\ \vdots \\ y_m \end{bmatrix}, \tag{4}$$

$$\mathbf{Xtr} = \begin{bmatrix} \mathbf{Xtr}_1^{\mathsf{T}} \\ \mathbf{Xtr}_2^{\mathsf{T}} \\ \vdots \\ \mathbf{Xtr}_m^{\mathsf{T}} \end{bmatrix} = \begin{bmatrix} 1 & Xtr_{11} & \cdots & Xtr_{1n} \\ 1 & Xtr_{21} & \cdots & Xtr_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & Xtr_{m1} & \cdots & Xtr_{mn} \end{bmatrix}, \tag{5}$$

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_m \end{bmatrix}, \tag{6}$$

$$\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_m \end{bmatrix}. \tag{7}$$

Currently, in our developed code we do not use a constant term α_0 within the linear regression algorithm that is incorporated and used during the training and testing. To be consistent with the latest developments of ML models and avoid numerical instabilities, the constant term α_0 was excluded. An automatic check of the rank of the input matrix is then performed, and in the case where the matrix is found not to be a full rank, the collinear predictors are excluded.

Hence, we have a full column rank, and we can solve the system by utilizing the following equation:

$$\boldsymbol{\alpha} = (\mathbf{Xtr}^T \mathbf{Xtr})^{-1} \mathbf{Xtr}^T \mathbf{ytr} \tag{8}$$

Now, the R^2 parameter can be computed, by calculating the mean value of ytr:

$$\bar{ytr} = \frac{1}{m} \sum_{i=1}^{m} ytr_i \tag{9}$$

The sum of squares of the errors (residuals) is computed as:

$$SS_{res} = \sum_{i=1}^{m} (ytr_i - \hat{ytr}_i)^2 = \sum_{i=1}^{m} \boldsymbol{\varepsilon}_i^2 \tag{10}$$

and the total sum of squared errors of the target variables derives from its mean value

$$SS_{tot} = \sum_{i=1}^{m} (ytr_i - \bar{ytr})^2, \tag{11}$$

and hence

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}} \tag{12}$$

For more details and code information related to the linear regression implementation, refer to the code in file **ml_linear_regression.py**, and Section 18 (Appendix: Train Linear Regression - ml_linear_regression.py).

6.1.3 Logistic Regression

When the target variable is binary, taking values in the domain 0 or 1, we may fit a logistic function to the data, instead of a Linear Model. In this case, Equation 1 is transformed to:

$$ytr_i = \frac{1}{1 + e^{-(\alpha_0 + \alpha_1 Xtr_{i1} + \cdots + \alpha_n Xtr_{in})}}. \tag{13}$$

this is based on the logistic function

$$y(x) = \frac{1}{1 + e^{-(\alpha \times x + \beta)}}. \tag{14}$$

In order to solve Equation 14 for the unknown weights α and β, we may write

$$1 + e^{-(\alpha \times x + \beta)} = \frac{1}{y}, \tag{15}$$

$$e^{-(\alpha \times x + \beta)} = \frac{1}{y} - 1, \tag{16}$$

$$\frac{1}{e^{(\alpha \times x + \beta)}} = \frac{1 - y}{y}, \tag{17}$$

$$\frac{1}{e^{(\alpha \times x + \beta)}} = \frac{1 - y}{y}, \tag{18}$$

$$e^{(\alpha \times x + \beta)} = \frac{y}{1 - y}, \tag{19}$$

and hence

$$\alpha \times x + \beta = \ln \frac{y}{1 - y}. \tag{20}$$

The function $\ln \frac{y}{1-y}$ is named *logit* function. Accordingly, we may solve for the weights α and β, in the form of a linear system, where the target variable, is the $logit(y)$.

6.2 Polynomial Regression (POLYREG-HYT)

Figure 9 shows the part of the code that has to be executed (select the part of this code with your mouse and then from your keyboard Shift+Enter) to perform the train and test with the polynomial regression method with hyperparameter tuning (POLYREG-HYT). In this command, we perform Polynomial Least Squares fitting [12, 45, 48, 49]. You may read also in [14] some underlying theoretical concepts, in Sections 1 & 2. The full method is developed and sent for publication, where the reference will be provided as soon as the manuscript is accepted for publication.

```
68  ## Polynomial Regression (POLYREG-HYT)
69  import import_libraries; reload(import_libraries)
70  import ml_nlregr; reload(ml_nlregr)
71  ml_nlregr.do_nlregr(Xtr, Xte, ytr, yte, features_names, target_name, LOG_REGR, do_permute)
72
```

Figure 9: Polynomial Regression Code: Lines 69-71

According to the developed method, we have to select from a vast pool of potential nonlinear features, as well as their number, we use algorithm 1 for feature selection, which has been found to be very efficient after an extensive parametric investigation. Accordingly, we automatically extract a closed-form formula in text format and a corresponding formula that can be used directly within the MS Excel[3] dataset that the user defined for the model development (see Section 4).

6.2.1 Literature on Higher Order Models

Some material that is considered valuable when trying to understand how this type of methods work is provided herein:

- Polynomial Least Squares Fitting [45]. Studying the mathematical part is optional.
- Polynomial Regression with Multiple Variables [12, 48, 49]
- (Optional) Calculus Underlying Theory [14]. Read Sections 1, 2.

6.2.2 Feature Selection Algorithm

As it was stated above, selecting the features during the training with POLYREG-HYT is of significant importance, therefore, the implementation of a feature selection algorithm is crucial for the development of an optimum predictive formula. As presented in the Algorithm, let p be the total number of polynomial features, with m_f representing the maximum number of formula features. The proposed algorithm aims to identify the indices $[o] = \{o_1, o_2, \ldots, o\} \subset [p] = \{1, 2, \ldots, p\}$ for minimizing the regression error e_i during the loop over iteration i. It must be noted here that $\mathbf{X}$ is the full input matrix, and $\mathbf{X}'$ is the corresponding input matrix with $[o]$ columns only.

For more information about the code, the reader is refered to file **ml_nlregr.py**, and Section 19 (Appendix: Train Polynomial Regression - ml_nlregr.py).

6.3 XGBoost-HYT-CV

We use XGBoost [30] with hyperparameter tunning (HYT), and cross-validation (CV). The reference on the complete method, XGBoost-HYT-CV will be provided here soon. According to the proposed methodology, we may define the number of folds, as well as the train-validation split percentage. The tunning is performed for the parameters and default values, where the user can also change and re-defined the values of these parameters through the software's interface:

1. The Maximum Number of XGBoost-HYT-CV Rounds

[3] https://www.microsoft.com/fi-fi/microsoft-365/excel

Algorithm 1: Polynomial Feature Selection Algorithm of POLYREG-HYT

Data: $\mathbf{X}, \mathbf{y}, m_f$ (maximum number of features)
Result: Initialize $[o] = 1$ with the constant term $\in [p]$
Solve Linear System $\mathbf{X}' \times \mathbf{a} = \mathbf{y}$, where $\mathbf{X}' \subset \mathbf{X}$, with $[o]$ columns.
Compute regression errors e_1.
Set as optimal error $\hat{e} \leftarrow e_1$.
Set as optimal indices $[\hat{o}] \leftarrow [o]$.
for $i \in [1, 2, \ldots, l]$ **do**
  repeat
    Select an index $d \in [p]$ randomly.
    if $d \in [o]$ **then**
      $r \leftarrow \mathcal{U}(0, 1)$
      if $r < \frac{1}{2}$ **then**
        Select randomly $o_d \in [p] : o_d \notin [o]$
        $[o] \leftarrow ([o] \setminus d) \cup o_d$;
      else
        $[o] \leftarrow [o] \setminus d$;
      end
    else
      if $o < m_f$ **then**
        $[o] \leftarrow [o] \cup d$;
      else
        Select randomly $o_d \in [o]$
        $[o] \leftarrow ([o] \setminus o_d) \cup d$;
      end
    end
  until $rank(\mathbf{X}') \equiv o$;
  Solve Linear System $\mathbf{X}' \times \mathbf{a} = \mathbf{y}$.
  Compute regression error e_i.
  if $e_i < \hat{e}$ **then**
    $\hat{e} \leftarrow e_i$
    $[\hat{o}] \leftarrow [o]$
  else
    $[o] \leftarrow [\hat{o}]$
  end
end

2. The Maximum tree depth $\in [1, 7, 15]$
3. The parameter eta $\in [0.05, 0.2, 0.5]$
4. The parameter colsample_bytree $\in [0.5, 1]$, and
5. The parameter subsample $\in [0.5, 1]$

The reader is advised to refer to the code of the software that is found within file **ml_xgboost.py**, and Section 20 (Appendix: Train XGBoost - ml_xgboost.py).

6.4 Random Forests (RF-HYT)

Random Forests with hyperparameter tunning (RF-HYT) is incorporated within the software, which is able to execute the corresponding ML algorithm through the lines of code shown in Figure 10.

```
81  ## Random Forests (RF-HYT)
82  import ml_random_forests; reload(ml_random_forests)
83  __thres_early_stop__ = 1e-2
84  __thres_min_tune_rounds__ = 100
85  ml_random_forests.do_random_forests(Xtr,Xte,ytr,yte,features_names,target_name,__thres_early_stop__,__thres_min_tune_rounds_
86
```

Figure 10: Random Forests Code: Lines 82-85

The respective code related to the RF-HYT ML algorithm can be found in file **ml_random_forests.py**, and Section 21 (Appendix: Train Random Forests - ml_random_forests.py).

6.5 Artificial Neural Networks

In order to execute the Artificial Neural Network (ANN) algorithm, the code that is shown in Figure 11 has to be selected and executed. This algorithm has the ability to automatically develop the neural network and train for the optimum predictive model [1].

```
88  ## ANNBN
89  reload(import_libraries); reload(misc_functions)
90  import ml_ANNBN; reload(ml_ANNBN)
91  ml_ANNBN.do_ANNBN(Xtr, Xte, ytr, yte, features_names, target_name, do_permute, 20, LOG_REGR)
92
```

Figure 11: ANNBN Code: Lines 89-91

The method presented in [1] is utilised herein to construct an ANN. Particularly, the inner layer's neurons weights $\mathbf{w}_k$ is computed, which is equivalent to approximating the problem by

$$\begin{pmatrix} \sigma(x_{11k}w_{1k} + x_{12k}w_{2k} + \cdots + x_{1nk}w_{nk} + b_k) \\ \sigma(x_{21k}w_{1k} + x_{22k}w_{2k} + \cdots + x_{2nk}w_{nk} + b_k) \\ \vdots \\ \sigma(x_{m_k1k}w_{1k} + x_{m_k2k}w_{2k} + \cdots + x_{m_knk}w_{nk} + b_k) \end{pmatrix} = \begin{pmatrix} y_{1k}, \\ y_{2k}, \\ \cdots, \\ y_{m_kk} \end{pmatrix}, \tag{21}$$

and hence,

$$\underbrace{\begin{pmatrix} x_{11k} & x_{12k} & \cdots & x_{1nk} & 1 \\ x_{21k} & x_{22k} & \cdots & x_{2nk} & 1 \\ \vdots & \vdots & \ddots & \vdots & 1 \\ x_{m_k1k} & x_{m_k2k} & \cdots & x_{m_knk} & 1 \end{pmatrix}}_{\mathbf{X}_k} \underbrace{\begin{pmatrix} w_{1k} \\ w_{2k} \\ \vdots \\ w_{nk} \\ b_k \end{pmatrix}}_{\mathbf{w}_k} = \underbrace{\begin{pmatrix} \sigma^{-1}(y_{1k}) \\ \sigma^{-1}(y_{2k}) \\ \vdots \\ \sigma^{-1}(y_{m_kk}) \end{pmatrix}}_{\hat{\mathbf{y}}_k}. \tag{22}$$

For those that would like to expand their knowledge related to ANN, it is recommended to read from [13], Sections 1 for important generic information, 2.1 for the basic formulation of ANNs, 3.1, 3.2, and 3.3 for Basic Applications of AI. Further reading can be found in [29].

For detailes related to the relevant code on can see file **ml_ANNBN.py**, and Section 22 (Appendix: Train ANNBN - ml_ANNBN.py).

6.6 Parallel Deep Learning ANN with Hyperparameter Tunning

We use PyTorch Framework [41] with hyper-parameter tuning in ml_DANN.py. This developed AI algorithm is known as DANN-MPIH-HYT.

The code that is required to be executed so as to implement the DANN-MPIH-HYT is given in Figure 12.

```
94  ## Deep Learning (DANN-MPIH-HYT)
95  import ml_DANN; reload(ml_DANN)
96  __thres_early_stop__ = 1e-2
97  __thres_min_tune_rounds__ = 100
98  ml_DANN.do_DANN(Xtr,ytr,Xte,yte,features_names,target_name,__thres_early_stop__,__thres_min_tune_rounds__,do_permute,LOG_RE(
99
```

Figure 12: DANN-MPIH-HYT Code: Lines 95-98

For additional material related to ANNs, please refer to:

- See this video for distributed deep learning: Training Deep Neural Networks on Distributed GPUs (Clickable link).
- Deep Learning: An MIT Press book [42](Open Access).
- A visual proof that neural nets can compute any function (Clickable link).

To see the relevant DANN-MPIH-HYT code, please refer to the file **ml_DANN.py**, and Section 23 (Appendix: Train DANN - ml_DANN.py).

6.7 Optimization Reliability

When working with ANNBN, XGBoost-HYT-CV, and DANN-MPIH-HYT reliability of the optimization process is performed. This means that the obtained Train-Validation-Test curves, correspond to the performance of each one of the models, and they are sorted by their validation performance. The developed figures can be found within the folder that the dataset is found in.

6.8 Accuracy Metrics

It is well known that in order to evaluate the performance of any predictive model, specific tools have to be used that will allow for an objective evaluation of the numerical response of the derived model. In this case, the following error metrics are utilized:

- The Pearson Correlation Coefficient:

$$\rho_{y\hat{tr}_i, ytr} = \frac{\text{cov}(y\hat{tr}_i, ytr)}{\sigma_{y\hat{tr}_i}\sigma_{ytr}}, \tag{23}$$

where

$$\text{cov}(y\hat{tr}, ytr) = \frac{1}{m}\sum_{i=1}^{m}(y\hat{tr}_i - \frac{1}{m}\sum_{i=1}^{m} y\hat{tr}_i)(ytr_i - \frac{1}{m}\sum_{i=1}^{m} ytr_i), \tag{24}$$

and σ denotes the standard deviation,

- The Root Mean Squared Error

$$RMSE = \sqrt{\frac{\sum_{i=1}^{m}\left(y\hat{tr}_i - ytr_i\right)^2}{m}}, \tag{25}$$

- The Mean Absolute Error

$$MAE = \frac{\sum_{i=1}^{m}\left|y\hat{tr}_i - ytr_i\right|}{m} \tag{26}$$

- The Mean Absolute Percentage Error

$$MAPE = \frac{1}{m}\sum_{i=1}^{m}\frac{\left|y\hat{tr}_i - ytr_i\right|}{ytr_i} \tag{27}$$

- The Maximum Absolute Percentage Error

$$MAXAPE = \max \frac{|\hat{ytr}_i - ytr_i|}{ytr_i} \tag{28}$$

- The Mean Absolute Mean Percentage Error

$$MAMPE = \frac{1}{m} \sum_{i=1}^{m} \frac{|\hat{ytr}_i - ytr_i|}{\frac{1}{m} \sum_{i=1}^{m} ytr_i} \tag{29}$$

as well as

- The Slope of the Predicted vs Actual values α, such that

$$\hat{\mathbf{ytr}} = \alpha \times \mathbf{ytr} + \beta \tag{30}$$

.

7 Sensitivity Analysis

Presenting the numerically obtained results is of great importance, where the investigation of the derived predictive models and their ability to predict values of out-of-sample data is something that any data analyst should perform after the completion of the training and resting. Additionally, the investigation of the sensitivity of the derived predictive models is extremely important, especially when the user aims in developing models that will require the minimum number of input parameters. For this reason, the software foresees the execution of a sensitivity analysis that will allow the user to determine which parameters are the most important to be used when trying to predict targets.

As soon as the trained models are obtained, all the features within the dataset are kept constant at specific values (i.e. 25%, Median, and 75% Quantiles), and then the perturbation of a specific feature is performed, for its given values. Accordingly, we obtain the corresponding sensitivity curves for each feature and each one of the trained models can be evaluated according to the developed sensitivity curves.

Henceforth, the importance of each one of the features for the target variable is illustratively demonstrated, where the user can decide on whether to keep or discard a specific parameter. For example, if one performs the analysis of a dataset that deals with the prediction of the shear strength of RC beams, they will find that [8] the shear capacity is practically not affected by the concrete's Young modulus of elasticity. The same was found when investigating the steel rebar's Young modulus of elasticity. On the other hand, Markou and Bakas [8] found that the effective depth of the RC beams was the parameter with the highest sensitivity, which is also a finding that is confirmed by laboratory experiments. This demonstrates the abilities and significance of utilizing this tool after the development of any predictive model.

In order to execute the sensitivity analysis code, the user has to select the code shown in Figure 13 and Shit+Enter.

```
101  ######################################################################
102  # Sensitivity Analysis
103  reload(misc_functions)
104  misc_functions.gather_all_sensitivity_curves(Xtr, features_names, target_name)
105  ######################################################################
106
```

Figure 13: Sensitivity Analysis Code: Lines 103-104

8 Adequacy of Data

Furthermore, we can easily train iteratively for a partial random subset of the train set, starting e.g. from the 25% of the observations, up to the 100%, with 100 intermediate new training for all methods. Therefore, the adequacy of the data can be evaluated from the shape of the curve depicting the performance of each partial training. Modern demand for data-centric AI has to be considered, thus the algorithm for evaluating the impact of the dataset's volume on each model's performance is implemented when training, testing, and validating any predictive model.

9 Error Analysis

After the training of each Machine Learning Model, a folder named Error_Analysis is created, where we plot the number of observations per bin in the train set vs MAE in each bin to illustrate the dependence of test set errors on the train set observations frequency. In this Error_Analysis folder, one can also find the empirical cumulative distribution function (CDF) graph for each set (Train-Validation-Test).

10 Predict out-of-sample Data

At this stage, the development and investigation of the predictive models have been completed, whereas validation through the use of additional data can also be performed. Furthermore, a user might like to use the developed predictive models to compute the values by using data received from the field. Therefore, use the models to predict the fundamental period of a new structure or estimate the shear capacity of an RC beam that they investigate.

To perform the prediction for values that are either out-of-sample or just for numerical purposes that foresee the prediction of values by using other input data, the user has to highlight and execute the code shown if Figure 14. Select a new excel, with out-of-training data, to test the accuracy. Use the same format for features and target, as the basic input file and utilize the # Predict section.

Then the new dataset file should be placed within the Predict folder and the relevant code should be executed. It must be also noted here that the predictive formula that derives from the POLYREG-HYT method is already available, thus the polynomial regression method is not included in this block of code. The user can simply use the developed predictive formula within the dataset that is found in the .xlsx file they developed.

```
108  ######################################################################
109  # Predict
110  from importlib import reload
111  import misc_functions; reload(misc_functions)
112  Xout, yout, features_names, target_name = misc_functions.read_out_of_sample_data()
113  Xout,yout = misc_functions.delete_missing_values_rows(Xout,yout)
114  import ml_linear_regression; reload(ml_linear_regression)
115  ml_linear_regression.predict_linregr(Xout, yout, target_name, LOG_REGR)
116  import ml_xgboost; reload(ml_xgboost)
117  ml_xgboost.predict_xgboost(Xout, yout, target_name)
118  import ml_random_forests; reload(ml_random_forests)
119  ml_random_forests.predict_rf(Xout, yout, target_name)
120  import ml_ANNBN; reload(ml_ANNBN)
121  ml_ANNBN.predict_ANNBN(Xout, yout, target_name)
122  import ml_DANN; reload(ml_DANN)
123  ml_DANN.predict_DANN(Xout, yout, target_name)
124  ######################################################################
```

Figure 14: Predict Code: Lines 110-123

Before moving to the next section, where a numerical implementation will be presented through the use of the **nbml** software, it must be noted that the entire Python code is provided through Appendices that are found at the end of this document.

11 Numerical Example

This section of the document will be presenting numerical implementations through the use of the software, while discussing the results and where the user can find them.

11.1 Fundamental Period of Reinforced Concrete Structures with Soil-Structure Interaction

The dataset developed and published in [11] is used herein to demonstrate the execution of the software and how the user can get results from the different folders that will be created within the "datasets" folder. The structural problem in this case deals with the development of predictive models for the computation of the fundamental period of RC structures with soil-structure interaction (SSI). The dataset included the fundamental period of both fixed and SSI models.

Figure 15 shows the dataset location (Fundamental RC Structures 2022.xlsx) and path to the excel file. Also, the Excel can be seen consisting the 7 columns with the data related to the soil depth and Young modulus, the height of the structure, its length and width, and the ratio of the shear walls [11]. The first 6 columns comprise the input data (features) and the last column consists of the fundamental period values (target).

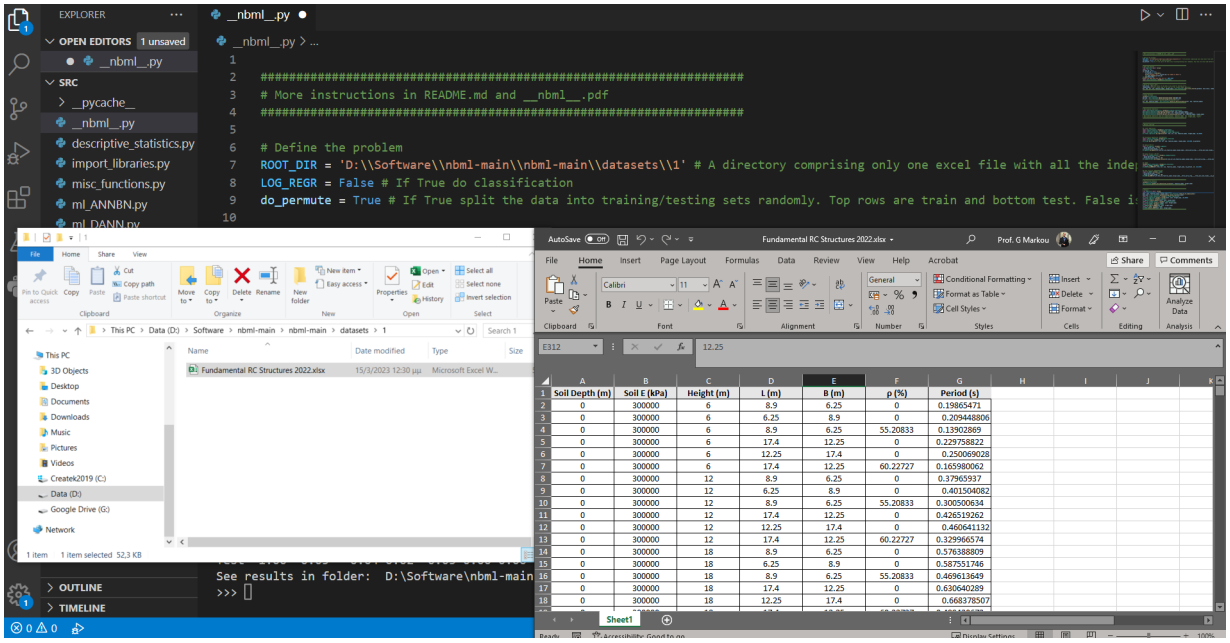


Figure 15: Dataset location and path to the excel file. Also, the Excel can be seen consisting the 7 columns with the data.

In addition to the above, in Figure 15 one can easily see the path that is defined within the code in Line 7 (D:\\Software\\nbml-main\\nbml-main\\datasets\\1). Figure 16 shows the first 3 lines of code that have been selected and executed by using the keys Shift+Enter on the keyboard. The corresponding messages that appear at the Terminal window are also seen in the same figure. The next step is to run the block that is responsible to open the dataset and check that our code allocates the Excel file successfully (see Figure 17). It is recommended that you select the lines of code by starting from the top line and selecting the last line of code that you want to execute including the line below making sure that the entire line of code is selected. In the case that you partially select a line and use Shift+Enter, then the line that was not selected as a whole will not be executed. This is shown in Figure 18 where the lines with comments were also selected prior to using Shift+Enter.

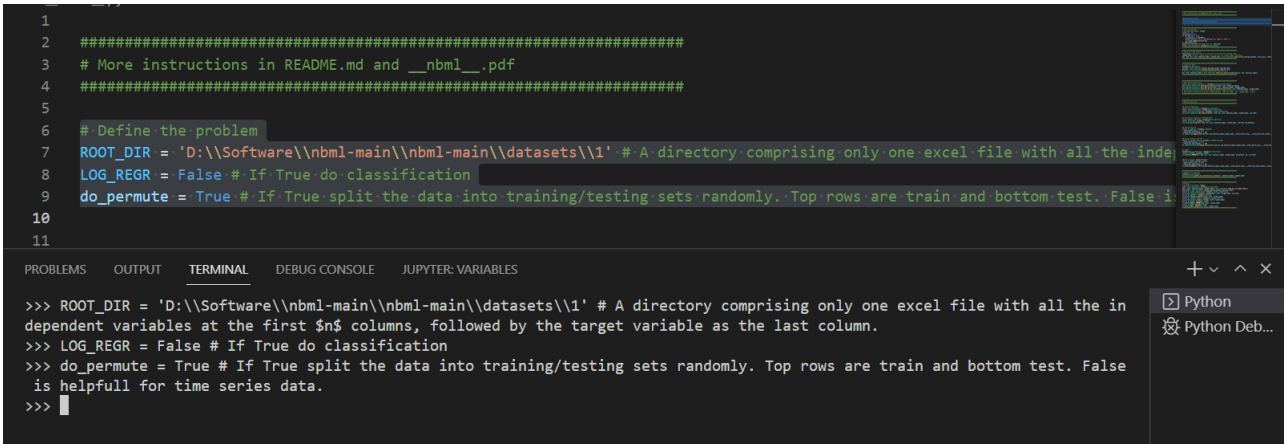


Figure 16: Selecting and running (Shift+Enter) the first 3 lines of code.

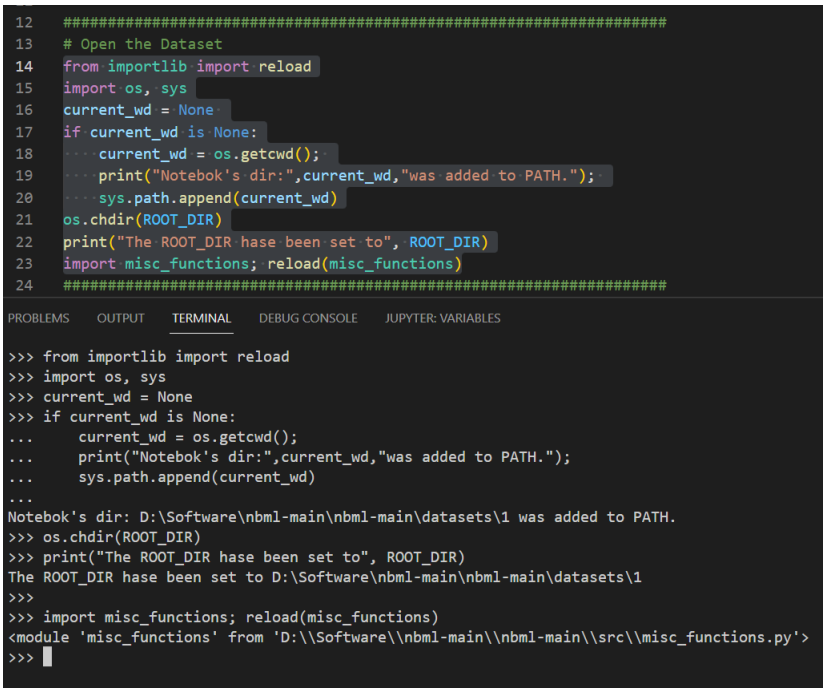


Figure 17: Selecting and running (Shift+Enter) the open the dataset code.

It is easy to observe that the Terminal window (see Figure 18) is now stating that the test ratio is 0.2 and that the training set size is 632. The test set is computed and set as 158. One can also note the the part of the comments that were selected "Split to Train & Test" were not executed since the syntax was not valid, thus the execution of that part of the code is disregarded by Visual Studio Code and the Python compiler.

The next step, as previously described in this document, is to perform the cleaning of the dataset, a procedure that can be seen in Figure 19. As it is shown in this figure, the algorithm found no missing values, where the identical rows command was not executed since it was chosen not to perform this check. This was done given that we know that this relatively small dataset does not have any identical rows. The relevant warning messages can be seen in Figure 19 within the Terminal window.

```
26  ####################################################################
27  # Split to Train & Test
28  test_ratio = 0.2 #The ratio of the data to be used for testing (0-1).
29  random_seed = 0 #The random seed to be used for the random number generator.
30  Xtr, Xte, ytr, yte, features_names, target_name = misc_functions.split_train_test(do_permute, test_ratio, random_seed)
31  ####################################################################
32

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  JUPYTER: VARIABLES

>>> print("The ROOT_DIR hase been set to", ROOT_DIR)
The ROOT_DIR hase been set to D:\Software\nbml-main\nbml-main\datasets\1
>>>
>>> import misc_functions; reload(misc_functions)
<module 'misc_functions' from 'D:\\Software\\nbml-main\\nbml-main\\src\\misc_functions.py'>
>>> Split to Train & Test
  File "<stdin>", line 1
    Split to Train & Test
          ^
SyntaxError: invalid syntax
>>> test_ratio = 0.2 #The ratio of the data to be used for testing (0-1).
>>> random_seed = 0 #The random seed to be used for the random number generator.
>>> Xtr, Xte, ytr, yte, features_names, target_name = misc_functions.split_train_test(do_permute, test_ratio, random_seed)
Fundamental RC Structures 2022.xlsx is being used for training and testing.
Training set size:  632 Test set size:  158 Random Perumte = True , Random Seed = 0
>>> ############
>>>
>>>
```

Figure 18: Selecting and running (Shift+Enter) the split to train & test code.

```
34  ####################################################################
35  # Clean the Data
36  reload(misc_functions)
37  Xte,yte = misc_functions.delete_missing_values_rows(Xte,yte)
38  Xtr,ytr = misc_functions.delete_missing_values_rows(Xtr,ytr)
39  # Xtr,ytr = misc_functions.delete_identical_rows(Xtr,ytr)
40  Xtr, Xte, features_names = misc_functions.check_fix_multicolinearity(Xtr, Xte, features_names)
41  ####################################################################
42
43

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  JUPYTER: VARIABLES

>>> ####
>>>
>>> reload(misc_functions)
<module 'misc_functions' from 'D:\\Software\\nbml-main\\nbml-main\\src\\misc_functions.py'>
>>> Xte,yte = misc_functions.delete_missing_values_rows(Xte,yte)
Number of missing values:  0 :: []
>>> Xtr,ytr = misc_functions.delete_missing_values_rows(Xtr,ytr)
Number of missing values:  0 :: []
>>> # Xtr,ytr = misc_functions.delete_identical_rows(Xtr,ytr)
>>>
>>> Xtr, Xte, features_names = misc_functions.check_fix_multicolinearity(Xtr, Xte, features_names)
Rank of Xtr:  6  out of  6  features
No multicolinearity detected.
>>>
```

Figure 19: Selecting and running (Shift+Enter) the clean the data code.

```
45  ####################################################################
46  # Descriptive Statistics
47  import descriptive_statistics; reload(descriptive_statistics)
48  descriptive_statistics.descriptive_statistics(Xtr, Xte, features_names)
49  descriptive_statistics.plot_pdf_cdf_all(Xtr, ytr, features_names, target_name)
50  descriptive_statistics.plot_all_by_all_correlation_matrix(Xtr, ytr, features_names, target_name)
51  # descriptive_statistics.plot_all_timeseries(Xtr, features_names, ytr, target_name, "Train")
52  # descriptive_statistics.plot_all_timeseries(Xte, features_names, yte, target_name, "Test")
53  ####################################################################

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  JUPYTER: VARIABLES

>>>
>>> import descriptive_statistics; reload(descriptive_statistics)
<module 'descriptive_statistics' from 'D:\\Software\\nbml-main\\nbml-main\\src\\descriptive_statistics.py'>
>>> descriptive_statistics.descriptive_statistics(Xtr, Xte, features_names)
D:\Software\nbml-main\nbml-main\src\descriptive_statistics.py:50: RuntimeWarning: invalid value encountered in divide
  df = pd.DataFrame({'mean':100*(mean1-mean2)/mean1, 'median':100*(median1-median2)/median1, 'std':100*(std1-std2)/std1,
D:\Software\nbml-main\nbml-main\src\descriptive_statistics.py:51: RuntimeWarning: invalid value encountered in divide
  'min':100*(min1-min2)/min1, 'max':100*(max1-max2)/max1, 'skewness':100*(skewness1-skewness2)/skewness1,
Descriptive statistics saved in Descriptive_Statistics/descriptive_statistics_train_test_and_differences.xlsx
>>> descriptive_statistics.plot_pdf_cdf_all(Xtr, ytr, features_names, target_name)
PDF and CDF saved in Descriptive_Statistics/PDF_CDF
>>> descriptive_statistics.plot_all_by_all_correlation_matrix(Xtr, ytr, features_names, target_name)
All Features vs Target saved in Descriptive_Statistics/All_Features_vs_Target.png
6999 Map is ready, Optimal Objective:  0.9999996068702091
>>>
```

Figure 20: Selecting and running (Shift+Enter) the descriptive statistics code.

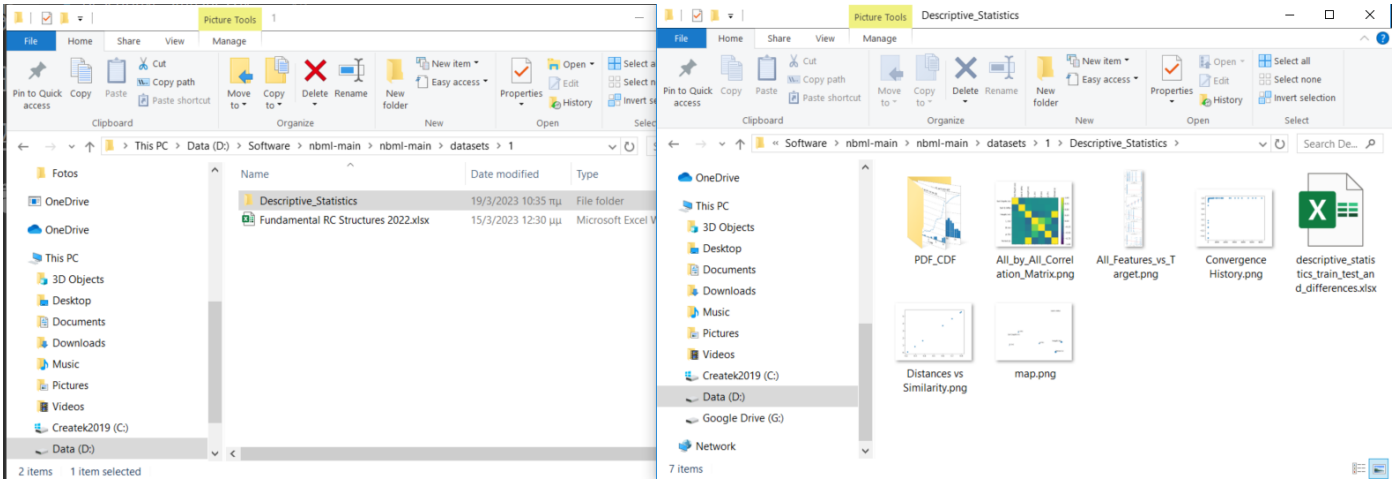


Figure 21: Files location and type of files generated by the descriptive statistics code.

After the execution of the clean data block of code, the software requires the execution of the descriptive statistics block of code that will generate all the required information about the dataset that was uploaded to the RAM and it is going to be used for the training and testing of the predictive models. Figure 20 shows the execution of the code related tot he descriptive statistics and the Terminal window warning messages. Furthermore, Figure 21 shows the location (left) and the actual set of files (right) that are generated after the execution of this part of the code.

The list of .png files and their titles that are generated by the software and the code developed under that block that deals with the descriptive statistics are the following:

1. All_by_All_Correlation_Matrix.png
2. All_Features_vs_Target.png
3. Convergence History.png
4. Distances vs Similarity.png
5. map.png

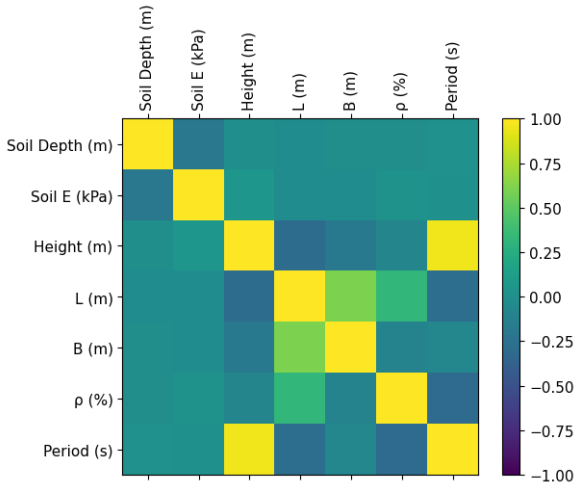


Figure 22: Correlation matrix for the fundamental period of RC structures dataset.

In this case, Figure 22 shows the resulted correlation matrix as it resulted fro all independent features found within the dataset, where Figure 23 shows a snapshot of the Excel table that is developed automatically and found at the same folder location as the .png files. Additionally, a folder named "PDF_CDF" is created as can be seen in Figure 21, were all the graphs related to the cumulative distribution functions can be found. In this case, a graph for each input feature is developed according to the values found within the dataset.

	mean	median	std	min	max	skewness	kurtosis
Soil Depth (m)	23.80032	22.5	20.12838	0	60	0.685244	-0.84194
Soil E (kPa)	11252642	300000	56114785	65000	3E+08	4.951218	22.51509
Height (m)	15.62658	18	9.551482	3	30	0.144732	-1.35711
L (m)	13.09304	12.25	6.758738	3.4	34.4	1.369892	2.059705
B (m)	11.86551	12.25	4.989755	3.4	34.4	0.879039	1.717037
ρ (%)	22.23969	0	32.79419	0	85.29412	0.931437	-0.9302

Figure 23: Snapshot of the Excel table presenting the statistics developed according to the values found within the dataset.

```
59  ######################################################################
60
61  ## Linear Regression
62  import misc_functions; reload(misc_functions)
63  import import_libraries; reload(import_libraries)
64  import ml_linear_regression; reload(ml_linear_regression)
65  ml_linear_regression.do_regression(Xtr, Xte, ytr, yte, features_names, target_name, LOG_REGR)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  JUPYTER: VARIABLES

<module 'misc_functions' from 'D:\\Software\\nbml-main\\nbml-main\\src\\misc_functions.py'>
>>>
>>> import import_libraries; reload(import_libraries)
<module 'import_libraries' from 'D:\\Software\\nbml-main\\nbml-main\\src\\import_libraries.py'>
>>>
>>> import ml_linear_regression; reload(ml_linear_regression)
<module 'ml_linear_regression' from 'D:\\Software\\nbml-main\\nbml-main\\src\\ml_linear_regression.py'>
>>> ml_linear_regression.do_regression(Xtr, Xte, ytr, yte, features_names, target_name, LOG_REGR)
Preparing Sensitivity Curves for feature: 1 of 6.
Preparing Sensitivity Curves for feature: 2 of 6.
Preparing Sensitivity Curves for feature: 3 of 6.
Preparing Sensitivity Curves for feature: 4 of 6.
Preparing Sensitivity Curves for feature: 5 of 6.
Preparing Sensitivity Curves for feature: 6 of 6.
Plotting Sensitivity Curves for feature: 1 of 6.
Plotting Sensitivity Curves for feature: 2 of 6.
Plotting Sensitivity Curves for feature: 3 of 6.
Plotting Sensitivity Curves for feature: 4 of 6.
Plotting Sensitivity Curves for feature: 5 of 6.
Plotting Sensitivity Curves for feature: 6 of 6.
         R  MAPE  MAMPE  MAE  RMSE  Acc  Sec
Train 0.99  0.14   0.08 0.04  0.05 0.00 0.06
Test  0.99  0.14   0.08 0.04  0.05 0.00 0.00
See results in folder:  D:\Software\nbml-main\nbml-main\datasets\1\ML_Models\LinRegr
>>>
```

Figure 24: Selecting and running (Shift+Enter) the Linear Regression ML code.

At this stage of the code, it is time to start training our first predictive model. As can be seen in Figure 24, the Linear Regression ML algorithm is executed and the respective warning messages are seen in the Terminal window of the Visual Code software. It is easy to observe that the software developed 6 sensitivity curves, according to the messages within the Terminal window (see Figure 24), while the basic information about the train and test error metrics are also provided.

It is also important to note here that the Terminal window informs the user that the results from the ML algorithm were generated and stored within a specific folder named LinRegr, where the exact comment that can be seen in Figure 24 reads: "See results in folder: D:\Software\nbml-main\nbml-main\datasets\1\ML_Models\LinRegr". Figure 25, shows the set of files that are generated within the LinRegr folder, which consists additional folders where the graphs and Excel Files are placed after the training is successfully performed.

In we check the files and folders shown in Figure 25, it is easy to observe that LinRegr is found within a folder named ML_Models, a folder that is created by the software and were all results are stored. Within the

ML_Models folder, two files can be seen. The first is an Excel file that contains all the ML related metrics that derive from all the training that were performed up to this point. Meaning that currently, only the Linear Regression metrics are stored within this file, whereas if we execute the training of a new predictive model by using the next ML algorithm (in this case the Polynomial Regression (POLYREG-HYT), then the software will take the new metrics and add them to this file (All_ML_metrics.xlsx). Therefore, each time we create a new predictive model by using a new ML or AL algorithm found within the software, the All_ML_metrics.xlsx file is updated accordingly.

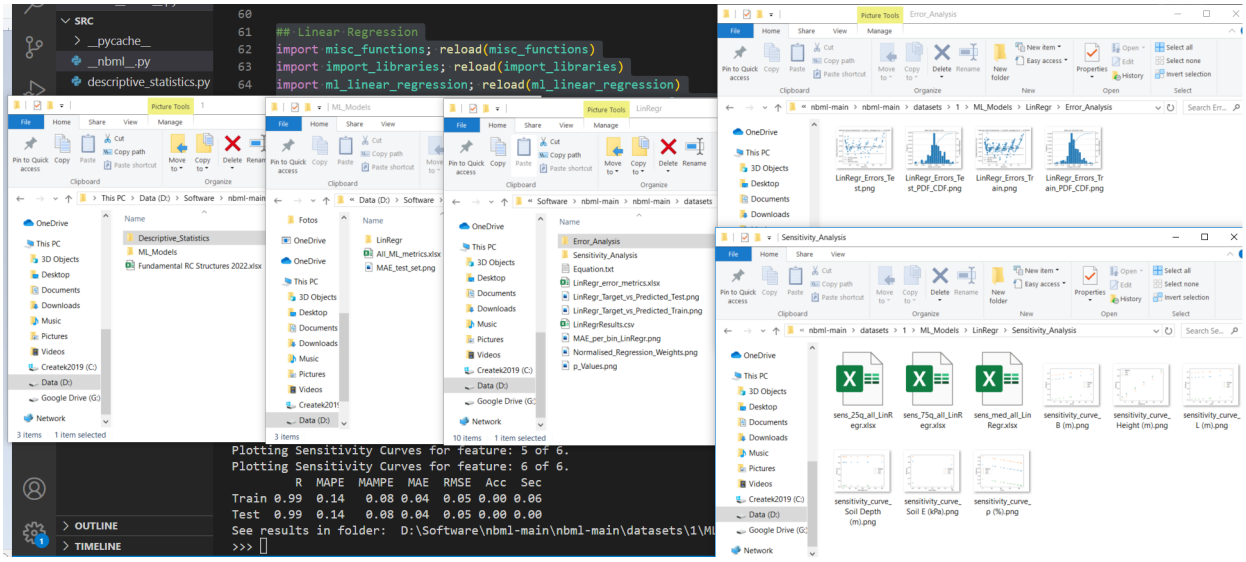


Figure 25: Files location and type of files generated by the Linear Regression code.

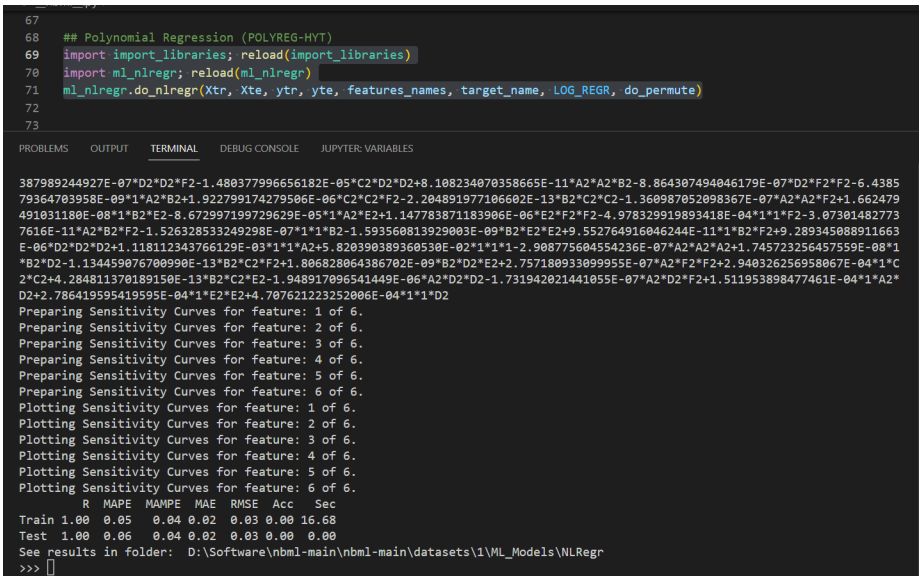


Figure 26: Files location and type of files generated by the Polynomial Regression code.

Let us proceed with the execution of the POLYREG-HYT method as shown in Figure 26. As can be seen, the software created a new folder named NLRegr, where the output has been saved. If we now open the All_ML_metrics.xlsx file and see the table that is created within, we will see what is shown in Figure 27. According to the developed table, the user has now all the required error metric information in a summarised

manner for both ML algorithms for training and testing datasets.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Model	Dataset	R	MAPE	MAMPE	MAE	RMSE	alpha	beta	Acc	FP	TP	FN	TN	Sec
2	LinRegr	Train	0.987307	0.135695	0.075299	0.036858	0.048705	0.969574	0.015884	0	0	0	0	0	0.061839
3	NLRegr	Train	0.996587	0.048735	0.038395	0.018794	0.025296	0.993185	0.003336	0	0	0	0	0	16.67758
4	LinRegr	Test	0.986872	0.144159	0.08117	0.037458	0.048788	0.97343	0.015249	0	0	0	0	0	0
5	NLRegr	Test	0.99603	0.055977	0.041591	0.019193	0.026852	0.992673	0.00416	0	0	0	0	0	0.000101

Figure 27: All_ML_metrics.xlsx file after executing the Linear and Polynomial Regression algorithms.

Furthermore, in the case of the POLYREG-HYT, a .txt file is created where the formula that can be used directly within the Excel dataset file to predict the fundamental period of RC structures can be found. Figure 28 shows the set of files that the software generated after the completion of the analysis with POLYREG-HYT, where the __equation_excel__.txt file can be also seen, which contains the Excel style formula. It is also important to note here that the development of the formula is performed in an automated manner, where the user does not need to define the number of features of the formula since the software is automatically developing the formula and its length in an optimum manner. Thus, the final length of the optimum predictive formula is performed in an automated manner by the software during the training procedure.

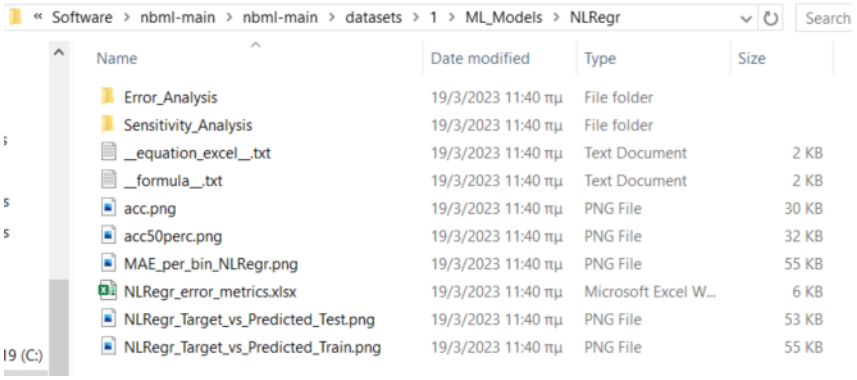


Figure 28: Output files found within the NLRegr folder after executing the Polynomial Regression algorithms.

It is important to note at this point that the developed formula found in the __formula__.txt file uses the names found within the first row of the Excel dataset file, while the parameters and names that are found in columns A-G are substituted be the respective cell location within the dataset and stored in __equation_excel__.txt file. This means that the first column of the dataset within cell A1 contains the same "Soil Depth (m)", which is used as the parameter name of the soil depth during the calculation of the fundamental period formula found in the __formula__.txt file. By replacing the name with the cell location A1, and by doing so for each and every input parameter, we get the corresponding formula found in the __equation_excel__.txt file.

At this point, we execute the remaining algorithms found within the code of our software and we derive the rest of the predictive models. In addition to that, the All_ML_metrics.xlsx file is update after each ML or AI algorithm is executed and the final table's snapshot is shown in Figure 29. According to the snapshot, the computational time in seconds is also provided for all methods for both train and test datasets. It is easy to observe that the fastest method that is also the worst in terms of accuracy is that of the Linear Regression. On the other hand, the most accurate method is that of RF-HYT followed by XGBoost-HYT-CV that derived a MAPE of 4.5% on the test dataset. The results are also indicating that XGBoost-HYT-CV derives a MAE that is slightly smaller than that of FR-HYT, but nevertheless, RF-HYT requires half the time to train compared to the XGBoost-HYT-CV. In terms of computational resources, the DANN-MPIH-HYT method is the one that requires the longest to train (997 seconds), which was expected since it is one of the main disadvantages of deep learning.

According to the obtained numerical results that are shown in Figure 29, it is easy to conclude that overall the RF-HYT is the optimum ML algorithm to train on this dataset. This of course is not always the case,

since according to the research work performed to date, it was found that the XGBoost-HYT-CV method is the one that outperforms all methods in terms of accuracy and objectivity when used to train different predictive models through the use of different datasets (March 2023). Now, before moving to the next stage of the software execution, it is important to note that the user's folder that includes all the resulted output files should look like Figure 30.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
1	Model	Dataset	R	MAPE	MAMPE	MAE	RMSE	alpha	beta	Acc	FP	TP	FN	TN	Sec
2	DANN	Train	0.993271	0.08488	0.056345	0.027581	0.037071	0.951884	0.023038	0	0	0	0	0	996.6409
3	LinRegr	Train	0.987307	0.135695	0.075299	0.036858	0.048705	0.969574	0.015884	0	0	0	0	0	0.061839
4	NLRegr	Train	0.996587	0.048735	0.038395	0.018794	0.025296	0.993185	0.003336	0	0	0	0	0	16.67758
5	RF	Train	0.998205	0.02839	0.026282	0.012865	0.018366	0.993982	0.002717	0	0	0	0	0	107.5575
6	XGBoost	Train	0.998148	0.030842	0.026699	0.013069	0.01864	0.996049	0.001934	0	0	0	0	0	197.8792
7	DANN	Test	0.992484	0.093508	0.061555	0.028407	0.038337	0.951506	0.024496	0	0	0	0	0	0.000399
8	LinRegr	Test	0.986872	0.144159	0.08117	0.037458	0.048788	0.97343	0.015249	0	0	0	0	0	0
9	NLRegr	Test	0.99603	0.055977	0.041591	0.019193	0.026852	0.992673	0.00416	0	0	0	0	0	0.000101
10	RF	Test	0.995029	0.039852	0.037787	0.017438	0.03024	0.978591	0.009037	0	0	0	0	0	0.005884
11	XGBoost	Test	0.995882	0.045318	0.037212	0.017173	0.027466	0.98363	0.006502	0	0	0	0	0	0.001396

Figure 29: All_ML_metrics.xlsx file after executing all ML and AI algorithms.

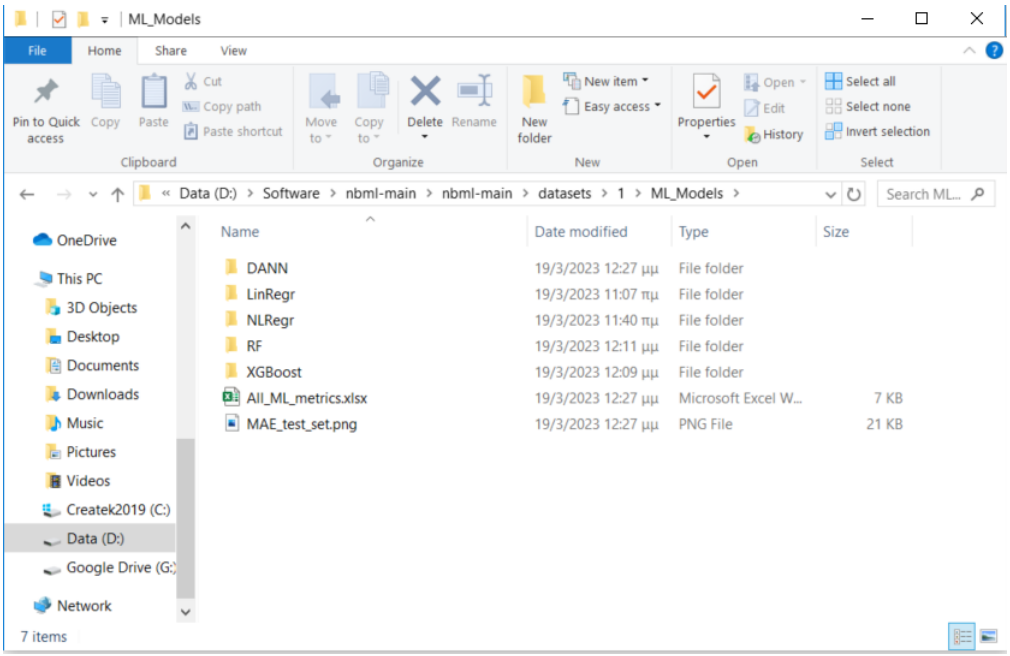


Figure 30: All folders after executing all available algorithms.

After the successful development of our predictive models, it is time to run the block of the code that will develop all the sensitivity curves and save them as .png files in a folder named ALL_Sensitivity which is found in folder ML_Models. These files can be seen in Figure 31, where for each input parameter found within the dataset the software creates comparative sensitivity curves according to the developed predictive model obtained from the different ML and AI algorithms. This will allow the user to verify the level of sensitivity for each feature and decide holistically if they will keep or remove a feature from next training that will aim to further optimize the numerically obtained predictive models.

The final step of this numerical implementation will discuss the prediction of values that are found within a new .xlsx file that has the exact same format as the initial dataset file. In our case herein the additional file named Validation2023.xlsx can be seen in Figure 32 and has the same columns and the input features are placed in the same order as the one found within the initial dataset file shown in Figure 15. As it can be seen in Figure 32, the validation file was chosen to consist of 5 rows that represent 5 new fundamental period calculations that are going to be used for validation purposes.

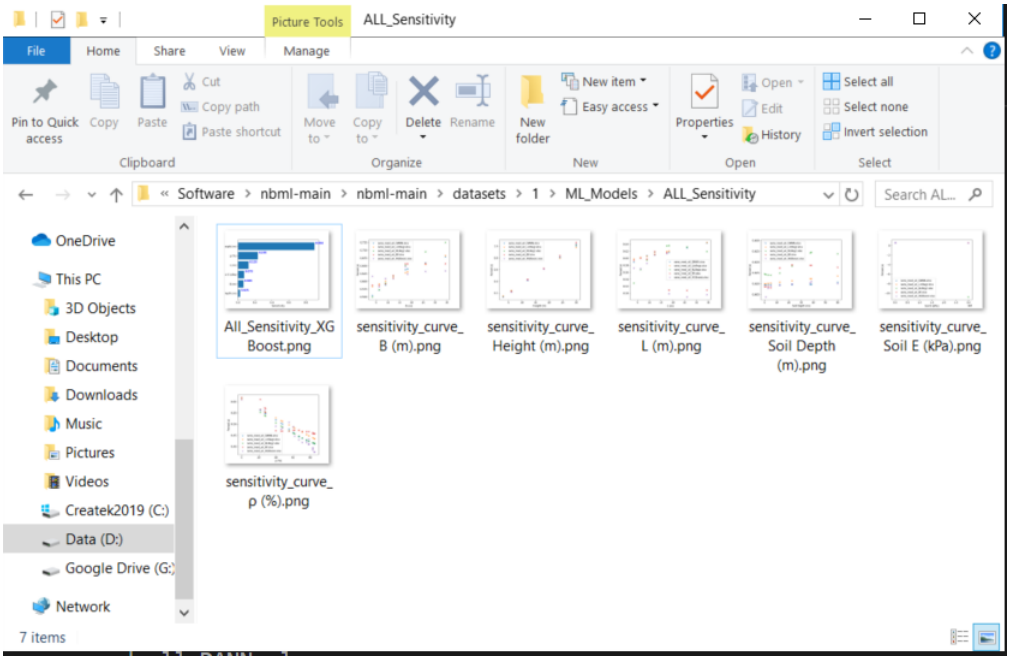


Figure 31: All sensitivity curve graph files.

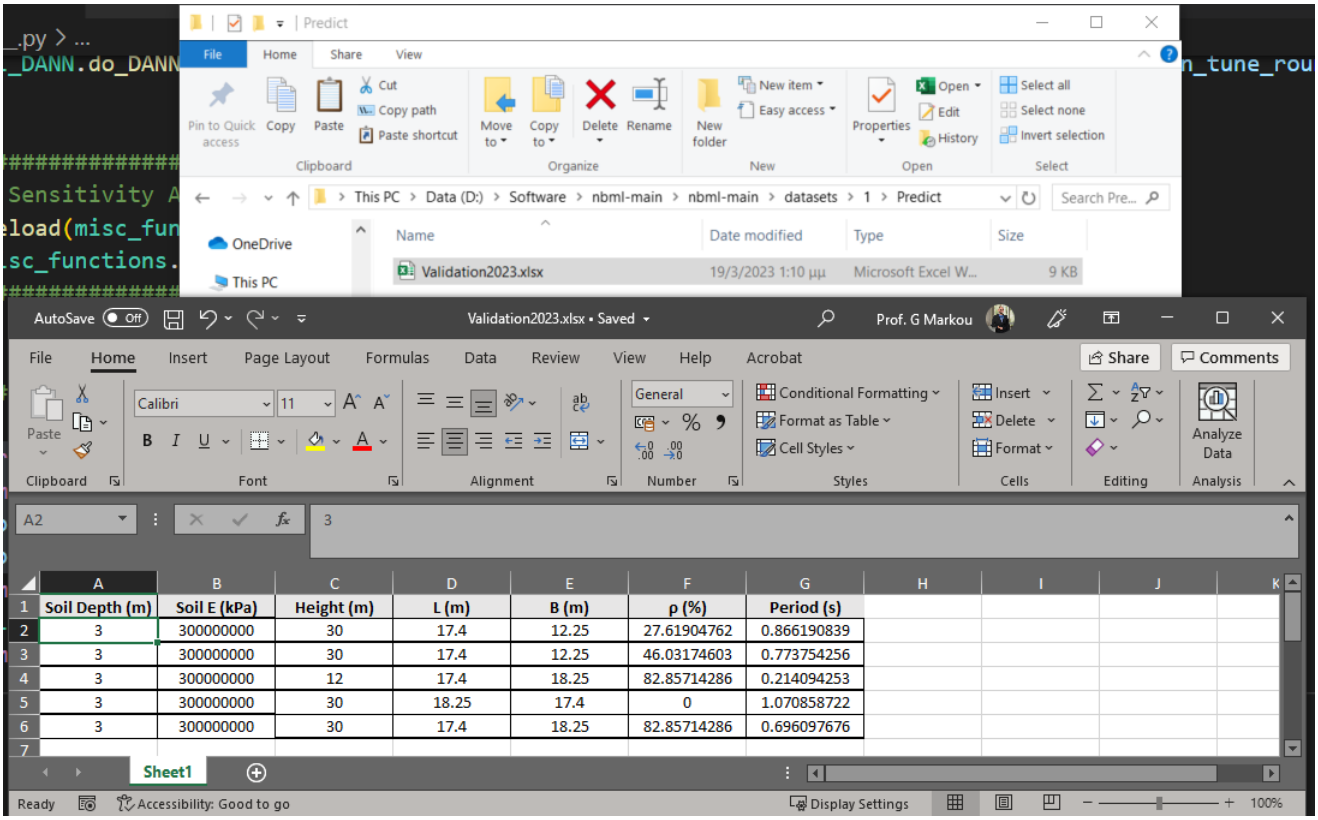


Figure 32: Location of Validation2023.xlsx file and the values found within this Excel file.

Figure 33 shows the selection of command lines 110-113 and their execution through using the keyboard keys Shit+Enter, and the respective Terminal messages. Figure 34 shows the files and folders that are

generated after command lines 116-117 are executed. These command lines refer to the use of the predictive model developed though the XGBoost-HYT-CV algorithm.

```
108  ######################################################################
109  # Predict
110  from importlib import reload
111  import misc_functions; reload(misc_functions)
112  Xout, yout, features_names, target_name = misc_functions.read_out_of_sample_data()
113  Xout,yout = misc_functions.delete_missing_values_rows(Xout,yout)
114  import ml_linear_regression; reload(ml_linear_regression)
115  ml_linear_regression.predict_linregr(Xout, yout, target_name, LOG_REGR)
116  import ml_xgboost; reload(ml_xgboost)
117  ml_xgboost.predict_xgboost(Xout, yout, target_name)
118  import ml_random_forests; reload(ml_random_forests)
119  ml_random_forests.predict_rf(Xout, yout, target_name)
120  import ml_ANNBN; reload(ml_ANNBN)
121  ml_ANNBN.predict_ANNBN(Xout, yout, target_name)
122  import ml_DANN; reload(ml_DANN)
123  ml_DANN.predict_DANN(Xout, yout, target_name)
124  ######################################################################
125

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE  JUPYTER: VARIABLES

>>>
>>> import misc_functions; reload(misc_functions)
<module 'misc_functions' from 'D:\\Software\\nbml-main\\nbml-main\\src\\misc_functions.py'>
>>> Xout, yout, features_names, target_name = misc_functions.read_out_of_sample_data()
No files exist in the directory
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: cannot unpack non-iterable NoneType object
>>> Xout,yout = misc_functions.delete_missing_values_rows(Xout,yout)
Number of missing values:  0 :: []
>>>
```

Figure 33: Executing the predict code for the import of the validation dataset.

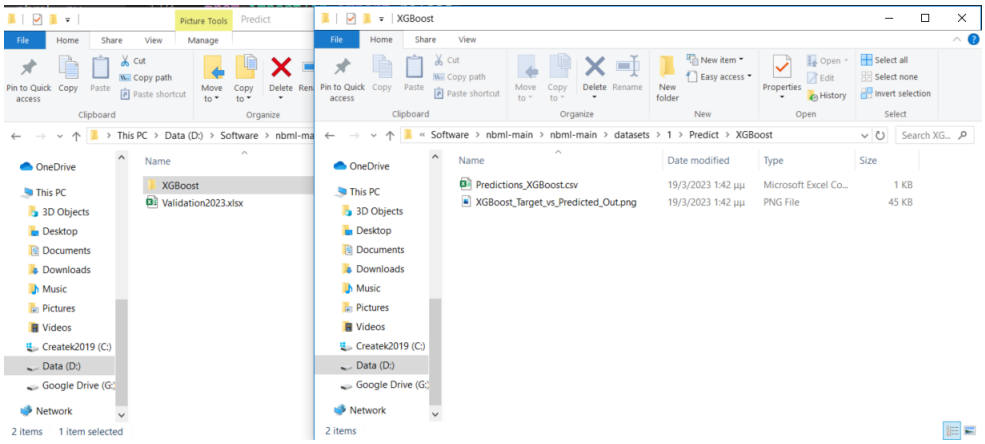


Figure 34: Files created after the validation dataset is used for the prediction of data through the use of the XGBoost-HYT-CV model.

As it can be seen in Figure 34, two files are created. The first is an Excel file named Predictions_XGBoost.csv that consists of the predicted values according to the model that was developed, while the second is a graph found within the file named XGBoost_Target_vs_Predicted_Out.png which can be seen in Figure 35. This graph compares the target vs predicted period values through which the user can determine the ability of their predictive model to capture out-of-sample data.

It is important to note here that regular updates will be performed to the code, therefore, additional ML algorithms and output in terms of graphs will be introduced in the near future. Therefore, if you note that the code is producing additional output files within the Statistics Descriptive, Error Analysis, etc., this is due to updates that are introduced to the code. The manual will be regularly updated, but not as frequently as the code.

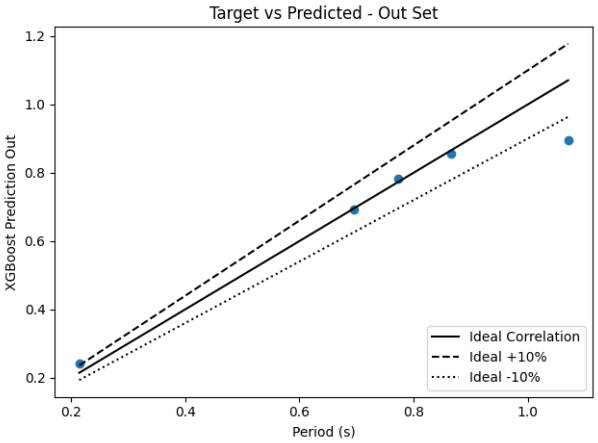


Figure 35: Comparison graph between the predicted fundamental periods through the use of the XGBoost-HYT-CV model and the values that were provided within the validation dataset.

12 Disclaimer

Although great effort has been made to create a bug-less software, due to the complexity of the calculations, as well as the high amount of statistical noise existing within the variables of a particular dataset, the predictions might deviate from the actual value of the target variable. These deviations in some cases might be high enough to disorientate the predictions and any corresponding decision-making. Furthermore, like in any predictive modelling case, past performance does not guarantee future results.

This user-manual, the corresponding software implementation, calculations, presented theory, and computer code, do not guarantee adequate accuracy, nor lack of errors, and should be utilized accordingly.

The authors will not accept liability for any loss or damage, including without limitation any loss of profit, which may arise directly or indirectly from the use of or reliance on such information and tools.

The software is provided "as is", without warranty of any kind, express or implied, including but not limited to the warranties of fitness for a particular purpose. In no event shall the authors or copyright holders be liable for any claim, damages or other liability arising from, out of or in connection with the software or the use or other dealings in the software.

We do not recommend the use of statistical analysis, machine learning and computational tools, as a sole means of industrial and scientific conclusions and decision making.

See also the repository's licence.

13 Acknowledgment

Parts of the code have been assisted with the GitHub Copilot https://github.com/features/copilot/ as well as ChatGPT https://chat.openai.com/chat created by OpenAI. Special thanks to GitHub, Microsoft for Visual Studio Code https://code.visualstudio.com/ and OpenAI.

14 References

[1] N. P. Bakas, A. Langousis, M. Nicolaou, and S. A. Chatzichristofis, "Gradient free stochastic training of anns, with local approximation in partitions," *Stochastic Environmental Research and Risk Assessment*, 2023. [Online]. Available: https://link.springer.com/article/10.1007/s00477-023-02407-2

[2] D. Koutsantonis, K. Koutsantonis, N. P. Bakas, V. Plevris, A. Langousis, and S. A. Chatzichristofis, "Bibliometric literature review of adaptive learning systems," *Sustainability*, vol. 14, no. 19, p. 12684, 2022. [Online]. Available: https://www.mdpi.com/2071-1050/14/19/12684/htm

[3] N. Bakas, D. Koutsantonis, V. Plevris, A. Langousis, and S. Chatzichristofis, "Inverse transform sampling for bibliometric literature analysis," in *The Thirteenth International Conference on Information, Intelligence, Systems and Applications. Ionian University, Corfu, Greece, 18-20 July 2022.* IISA 2022, 2022. [Online]. Available: http://easyconferences.eu/iisa2022/

[4] M. AlHamaydeh, G. Markou, N. Bakas, and M. Papadrakakis, "Ai-based shear capacity of frp-reinforced concrete deep beams without stirrups," *Engineering Structures*, vol. 264, p. 114441, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S014102962200551X

[5] N. Carstens, G. Markou, and N. Bakas, "Improved predictive fundamental period formula for reinforced concrete structures through the use of machine learning algorithms," in *14th International Conference on Agents and Artificial Intelligence*, February 2022. [Online]. Available: https://www.insticc.org/node/TechnicalProgram/icaart/2022/presentationDetails/109845

[6] E. Ababu, G. Markou, and N. Bakas, "Using machine learning and finite element modelling to develop a formula to determine the deflection of horizontally curved steel i-beams. international conference on agents and artificial intelligence," in *14th International Conference on Agents and Artificial Intelligence*, February 2022. [Online]. Available: https://www.insticc.org/node/TechnicalProgram/icaart/2022/presentationDetails/109824

[7] A. Westhuizen, G. Markou, and N. Bakas, "Development of a new fundamental period formula for steel structures considering the soil-structure interaction with the use of machine learning algorithms," in *14th International Conference on Agents and Artificial Intelligence*, 2022. [Online]. Available: https://www.insticc.org/node/TechnicalProgram/icaart/2022/presentationDetails/109784

[8] G. Markou and N. Bakas, "Prediction of the shear capacity of reinforced concrete slender beams without stirrups by applying artificial intelligence algorithms in a big database of beams generated by 3d nonlinear finite element analysis," *Computers and Concrete*, vol. 28, no. 6, 2021. [Online]. Available: http://www.techno-press.org/content/?page=article&journal=cac&volume=28&num=6&ordernum=1

[9] N. P. Bakas, V. Plevris, A. Langousis, and S. A. Chatzichristofis, "Itso: A novel inverse transform sampling-based optimization algorithm for stochastic search," *Stochastic Environmental Research and Risk Assessment*, 2021. [Online]. Available: https://link.springer.com/article/10.1007/s00477-021-02025-w

[10] V. Plevris, N. P. Bakas, and G. Solorzano, "Pure random orthogonal search (pros): A plain and elegant parameterless algorithm for global optimization," *Applied Sciences*, vol. 11, no. 11, 2021. [Online]. Available: https://www.mdpi.com/2076-3417/11/11/5053

[11] D. Gravett, C. Mourlas, V.-L. Taljaard, N. Bakas, G. Markou, and M. Papadrakakis, "New fundamental period formulae for soil-reinforced concrete structures interaction using machine learning algorithms and anns," *Soil Dynamics and Earthquake Engineering*, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0267726121000786

[12] T. Dimopoulos and N. Bakas, "Sensitivity analysis of machine learning models for the mass appraisal of real estate. case study of residential units in nicosia, cyprus," *Remote Sensing*, vol. 11, no. 24, p. 3047, 2019. [Online]. Available: https://www.mdpi.com/2072-4292/11/24/3047

[13] N. P. Bakas, A. Langousis, M. Nicolaou, and S. Chatzichristofis, "A gradient free neural network framework based on the universal approximation theorem," *arXiv preprint arXiv:1909.13563*, 2020. [Online]. Available: https://arxiv.org/abs/1909.13563

[14] N. P. Bakas, "Taylor polynomials in high arithmetic precision as universal approximators," *arXiv preprint arXiv:1909.13565*, 2019. [Online]. Available: https://arxiv.org/abs/1909.13565

[15] V. Plevris, G. Solorzano, and N. Bakas, "Literature review of historical masonry structures with machine learning," in *7th ECCOMAS Thematic Conference on Computational Methods in Structural Dynamics and Earthquake Engineering*. COMPDYN 2019, 2019. [Online]. Available: https://2019.compdyn.org/files/uploads/general/compdyn_2019_programme_online.pdf

[16] M. Papadaki, N. Bakas, E. Ochieng, I. Karamitsos, and R. Kirkham, "Big data from social media and scientific literature databases reveals relationships among risk management, project management and project success," in *6th Annual University of Maryland PM Symposium in May 2019*. pmworldjournal, 2019. [Online]. Available: https://pmworldjournal.com/article/big-data-from-social-media-and-scientific-literature-databases

[17] N. P. Bakas, "Numerical Solution for the Extrapolation Problem of Analytic Functions," *Research*, vol. 2019, pp. 1–10, may 2019. [Online]. Available: https://spj.sciencemag.org/research/2019/3903187/

[18] T. Dimopoulos and N. Bakas, "An artificial intelligence algorithm analyzing 30 years of research in mass appraisals," *RELAND: International Journal of Real Estate & Land Planning*, vol. 2, no. 0, pp. 10–27, 2019. [Online]. Available: http://ejournals.lib.auth.gr/reland/article/view/6749

[19] C. M. Faggion Jr, R. S. Ware, N. Bakas, and J. Wasiak, "An analysis of retractions of dental publications," *Journal of Dentistry*, vol. 79, pp. 19–23, dec 2018. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S0300571218304123

[20] T. Dimopoulos, H. Tyralis, N. P. Bakas, and D. Hadjimitsis, "Accuracy measurement of Random Forests and Linear Regression for mass appraisal models that estimate the prices of residential apartments in Nicosia, Cyprus," *Advances in Geosciences*, vol. 45, pp. 377–382, nov 2018. [Online]. Available: https://www.adv-geosci.net/45/377/2018/

[21] C. M. Faggion, N. P. Bakas, and J. Wasiak, "A survey of prevalence of narrative and systematic reviews in five major medical journals," *BMC Medical Research Methodology*, vol. 17, no. 1, p. 176, dec 2017. [Online]. Available: https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-017-0453-y

[22] N. Bakas, S. Makridakis, and M. Papadrakakis, "Torsional parameters importance in the structural response of multiscale asymmetric-plan buildings," *Coupled systems mechanics*, vol. 6, no. 1, pp. 55–74, mar 2017. [Online]. Available: http://www.techno-press.org/download.php?journal=csm&volume=6&num=1&ordernum=5

[23] J. Bellos, D. J. Inman, and N. Bakas, "Nature of coupling in non-conservative distributed parameter systems attached to external damping sources," *Mathematics and Mechanics of Solids*, p. 108128651771402, jun 2017. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1081286517714022

[24] S. Makridakis and N. Bakas, "Forecasting and uncertainty: A survey," *Risk and Decision Analysis*, vol. 6, no. 1, pp. 37–64, jan 2016. [Online]. Available: https://content.iospress.com/journals/risk-and-decision-analysis/6/1

[25] A. Kotsakis, M. Nübling, G. Pelekanakis, N. Bakas, and J. Thanopoulos, "The Greek COPSOQ v.3 Validation Study, a post crisis assessment of the Psychosocial Risks in Greece," in *2nd International Conference on Sustainable Employability - Building Bridges between Science and Practice*, 2018, p. 63. [Online]. Available: http://www.employability21.com/sites/default/files/uploads/ConferenceBookEmployability21.pdfhttp://www.employability21.com/sites/default/files/OS5-Kotsakis-GreekCOPSOQIII.pdf

[26] V. Plevris, N. Bakas, G. Markeset, and J. Bellos, "Literature review of masonry structures under earthquake excitation utilizing machine learning algorithms," in *Proceedings of the 6th International Conference on Computational Methods in Structural Dynamics and Earthquake Engineering (COMPDYN 2015)*. Athens: Institute of Structural Analysis and Antiseismic Research School of Civil Engineering National Technical University of Athens (NTUA) Greece, 2017, pp. 2685–2694. [Online]. Available: https://www.eccomasproceedia.org/conferences/thematic-conferences/compdyn-2017/5598

[27] N. Anastasi, N. Bakas, P. Tiano, J. Stuart, L. Wittig, J. Royo, L. Sanchez, O. Cuzman, and K. Richter, "EcoCement: A Novel Composite Material For The Construction Industry. Identification Of An Optimal Recipe Using Neural Networks," in *VII European Congress on Computational Methods in Applied Sciences and Engineering*, 2016. [Online]. Available: https://www.eccomas2016.org/proceedings/pdf/9472.pdf

[28] N. Bakas, J. Bellos, A. Kanyilmaz, and S. Makridakis, "Regression analysis vs genetic algorithms: computational efficiency assessment on the design of proindustry project SSDC isolators under Incremental dynamic loading," in *VII European Congress on Computational Methods in Applied Sciences and Engineering*, 2016. [Online]. Available: https://www.eccomas2016.org/proceedings/pdf/9475.pdf

[29] S. Chatzichristofis, N. Bakas*, and P. Razis, "High performance computing in cyprus," CERN - RECFA Meeting, University of Cyprus, 2019. [Online]. Available: https://indico.cern.ch/event/839488/timetable/#20191025

[30] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: ACM, 2016, pp. 785–794. [Online]. Available: http://doi.acm.org/10.1145/2939672.2939785

[31] S. Seabold and J. Perktold, "statsmodels: Econometric and statistical modeling with python," in *9th Python in Science Conference*, 2010.

[32] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[33] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, "API design for machine learning software: experiences from the scikit-learn project," in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, 2013, pp. 108–122.

[34] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[35] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[36] M. L. Waskom, "seaborn: statistical data visualization," *Journal of Open Source Software*, vol. 6, no. 60, p. 3021, 2021. [Online]. Available: https://doi.org/10.21105/joss.03021

[37] I. Flyamer, "adjusttext - automatic label placement for matplotlib." [Online]. Available: https://github.com/Phlya/adjustText

[38] J. D. Team, "nbconvert: Convert notebooks to other formats," 2015-2023. [Online]. Available: https://nbconvert.readthedocs.io/en/latest/index.html

[39] I. NumFOCUS, "pandas documentation," 2023. [Online]. Available: https://pandas.pydata.org/docs/index.html

[40] C. C. Eric Gazoni, "openpyxl - a python library to read/write excel 2010 xlsx/xlsm files," 2023. [Online]. Available: https://foss.heptapod.net/openpyxl/openpyxl

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

[42] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[43] E. W. Weisstein, "Least squares fitting," 2002. [Online]. Available: http://mathworld.wolfram.com/LeastSquaresFitting.html

[44] ——, "Correlation coefficient," 2002. [Online]. Available: http://mathworld.wolfram.com/CorrelationCoefficient.html

[45] ——, "Least squares fitting–polynomial," 2002. [Online]. Available: http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html

[46] D. Sculley, J. Snoek, A. Rahimi, and A. Wiltschko, "Winner's Curse? On Pace, Progress, and Empirical Rigor," *ICLR Workshop track*, 2018. [Online]. Available: https://openreview.net/forum?id=rJWF0Fywf

[47] M. Hutson, "AI researchers allege that machine learning is alchemy," *Science*, may 2018. [Online]. Available: http://www.sciencemag.org/news/2018/05/ai-researchers-allege-machine-learning-alchemy

[48] N. E. Helwig, "Regression with Polynomials and Interactions," 2017. [Online]. Available: http://users.stat.umn.edu/~helwig/notes/polyint-Notes.pdf

[49] J. Utts, "Polynomial and Interaction Models," 2013. [Online]. Available: https://www.ics.uci.edu/~jutts/201-F13/Lecture13.pdf

15 Appendix: Code to import_libraries.py

```
from adjustText import adjust_text

import codecs

from collections import namedtuple

import csv

import datetime

import glob

from importlib import reload

from itertools import product, combinations_with_replacement

import math

import matplotlib.pyplot as plt

from nbconvert import HTMLExporter
import nbformat

from numpy import amin, amax, arctanh, array, array_str, arange, argmax,
    argmin, argsort, column_stack
from numpy import c_, concatenate, copy, corrcoef, delete
from numpy import diag, digitize, divide, dot, empty, exp, fromiter, isinf,
    isnan
from numpy import Inf, linspace, loadtxt, log, logical_and, maximum, mean,
    median, newaxis, minimum, ndarray, newaxis, ones
from numpy import poly1d, polyfit, percentile
from numpy import quantile, repeat, roll, round, row_stack, savetxt, setdiff1d
    , sort, sqrt, std, sum, tanh, transpose, unique, where, zeros
from numpy.linalg import inv, lstsq, matrix_rank, solve, svd
from numpy.random import choice, rand, randint, permutation, seed
from numpy import append as np_append
from numpy import delete as np_delete
from numpy import round as np_round

import os

import pandas as pd

import pickle

import random

import seaborn as sns

from scipy.integrate import cumtrapz
from scipy.optimize import curve_fit
```

```
from scipy.stats import t, pearsonr, skew, kurtosis
from scipy.special import expit, logit
from scipy import linalg

from sklearn import datasets, linear_model
from sklearn.cluster import KMeans
from sklearn.linear_model import LinearRegression, QuantileRegressor
from sklearn.model_selection import GridSearchCV, train_test_split
from sklearn.utils import shuffle
from sklearn.metrics import mean_squared_error, r2_score, confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import MinMaxScaler
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeClassifier, plot_tree,
    DecisionTreeRegressor, export_text

import statsmodels.api as sm

import sys

from time import time

import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader

import xgboost as xgb
```

16 Appendix: Code for misc_functions.py

```
import os
from import_libraries import *

def create_ml_dir(ROOT_DIR, __method__):
    if not os.path.exists(ROOT_DIR+"ML_Models"+os.sep+__method__):
        os.makedirs(ROOT_DIR+"ML_Models"+os.sep+__method__)
    if not os.path.exists(ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
        Error_Analysis"):
        os.makedirs(ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
            Error_Analysis")
    if not os.path.exists(ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
        Sensitivity_Analysis"):
        os.makedirs(ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
            Sensitivity_Analysis")

def create_all_directories(ROOT_DIR, PERMUTE_TRAIN_TEST):
    if not os.path.exists(ROOT_DIR+"Descriptive_Statistics"):
        os.makedirs(ROOT_DIR+"Descriptive_Statistics")
    if not os.path.exists(ROOT_DIR+"Descriptive_Statistics"+os.path.sep+"
        PDF_CDF"):
        os.makedirs(ROOT_DIR+"Descriptive_Statistics"+os.path.sep+"PDF_CDF")
    if not os.path.exists(ROOT_DIR+"Descriptive_Statistics"+os.path.sep+"Tree"
        ):
        os.makedirs(ROOT_DIR+"Descriptive_Statistics"+os.path.sep+"Tree")
    if not os.path.exists(ROOT_DIR+"Descriptive_Statistics"+os.path.sep+"
        TimeSeries"):
        os.makedirs(ROOT_DIR+"Descriptive_Statistics"+os.path.sep+"TimeSeries"
            )
    if not PERMUTE_TRAIN_TEST:
        if not os.path.exists(ROOT_DIR + "Descriptive_Statistics" + os.sep + "
            TimeSeries"):
            os.makedirs(ROOT_DIR + "Descriptive_Statistics" + os.sep + "
                TimeSeries")

    __methods__ = ["LinRegr", "NLRegr", "XGBoost", "RF", "ANNBN", "DANN"]
    for __method__ in __methods__:
        create_ml_dir(ROOT_DIR, __method__)

    if not os.path.exists(ROOT_DIR + "ML_Models" + os.sep + "ALL_Sensitivity")
        :
        os.makedirs(ROOT_DIR + "ML_Models" + os.sep + "ALL_Sensitivity")

    if not os.path.exists(ROOT_DIR+"Predict"):
        os.makedirs(ROOT_DIR+"Predict")
    for __method__ in __methods__:
        if not os.path.exists(ROOT_DIR+"Predict"+os.sep+__method__):
            os.makedirs(ROOT_DIR+"Predict"+os.sep+__method__)


def regression_bak(X, Y):
    Result = namedtuple("Result", ["aa", "p_vals", "rr", "mean_resid", "
        t_statistic", "standard_errors", "residuals", "flag"])
```

```
    nn = len(Y)
    varss = X.shape[1]
    Xt = X.transpose()
    XtX = dot(Xt, X)
    try:
        invXtX = inv(XtX)
        aa = dot(dot(invXtX, Xt), Y)
        mean_y = mean(Y)
        sstot = sum((Y - mean_y) ** 2)
        predicted = dot(X, aa)
        residuals = Y - predicted
        ssreg = sum((predicted - mean_y) ** 2)
        ssres = sum(residuals ** 2)
        rr = 1 - ssres / sstot
        if rr < 1e-10000:
            return Result(aa, 0, rr, 0, 0, 0, 0, "rr")
        mean_resid = mean(residuals)
        ss_resid = sum((residuals - mean_resid) ** 2) / (nn - varss)
        var_covar = dot(invXtX, ss_resid)
        if min(diag(var_covar)) < 0:
            print("minimum(diag(var_covar))<0", min(diag(var_covar)))
            return Result(aa, 0, rr, 0, 0, 0, 0, "var_covar")
        standard_errors = sqrt(diag(var_covar))
        t_statistic = aa / standard_errors
        p_vals = 1 - t.cdf(abs(t_statistic), nn) + t.cdf(-abs(t_statistic), nn
            )
        return Result(aa, p_vals, rr, mean_resid, t_statistic, standard_errors
            , residuals, "OK")
    except Exception as ex1:
        print(ex1)
        return Result(0, 0, 0, 0, 0, 0, 0, ex1)


def split_train_test(PERMUTE_TRAIN_TEST, test_ratio, random_seed, ROOT_DIR):
    s_all = []
    for file in os.listdir(ROOT_DIR):
        if file.endswith(".xlsx"):
            s_all.append(ROOT_DIR + file)

    if len(s_all)>1:
        print("More than 1 files exists in the directory. Execution Stopped.")
        return
    else:
        my_file = s_all[0]
        print(my_file+" is being used for training and testing.")
    df = pd.read_excel(my_file)
    obs = df.shape[0]

    if PERMUTE_TRAIN_TEST:
        rng = random.Random()
        rng.seed(random_seed)
        shuffled_indices = rng.sample(list(range(obs)),obs)
    else:
        shuffled_indices = list(range(obs))
```

```
    test_set_size = int(obs * test_ratio)
    train_set_size = obs - test_set_size
    train_indices = shuffled_indices[:train_set_size]
    test_indices = shuffled_indices[train_set_size:]


    cols = df.columns
    df = df.to_numpy()

    # export_notebook_to_html()
    print("Training set size: ", train_set_size, "Test set size: ",
        test_set_size, "Random Perumte =", PERMUTE_TRAIN_TEST, ", Random Seed
        =", random_seed)

    return df[train_indices,:-1], df[test_indices,:-1], df[train_indices,-1],
        df[test_indices,-1], cols[:-1], cols[-1]

def plot_target_vs_predicted(ytr, pred, target_name, __method__, tag, path_):
    plt.scatter(ytr, pred)
    plt.xlabel(target_name)
    plt.ylabel(__method__.replace("_", " ") + " Prediction " + tag)
    plt.title("Target vs Predicted — "+tag+" Set")
    rr = linspace(min(ytr), max(ytr), 100)
    plt.plot(rr, rr, 'k', label="Ideal Correlation")
    plt.plot(rr, rr*1.1, 'k--', label="Ideal +10%")
    plt.plot(rr, rr*0.9, 'k:', label="Ideal -10%")
    plt.legend(loc='lower right')
    plt.tight_layout()
    plt.savefig(path_ + __method__ + "_Target_vs_Predicted_"+tag+".png")
    plt.close()

def error_metrics(ACT, PRED, LOGISTIC_REGR):
    _cor_ = corrcoef(ACT, PRED)[0,1]
    _mape_ = mean(abs(ACT-PRED)/abs(ACT))
    _mamte_ = mean(abs(ACT-PRED))/mean(abs(ACT))
    _mae_ = mean(abs(ACT-PRED))
    _rmse_ = sqrt(mean((ACT-PRED)**2))
    _beta_, _alpha_ = lstsq(column_stack((ones(len(ACT)), ACT)), PRED, rcond=
        None)[0]

    _accuracy_ = mean(ACT==round(PRED,0))
    # false positive rate
    _FP_ = mean((ACT==0) & (round(PRED,0)==1))
    _TP_ = mean((ACT==1) & (round(PRED,0)==1))
    _FN_ = mean((ACT==1) & (round(PRED,0)==0))
    _TN_ = mean((ACT==0) & (round(PRED,0)==0))

    if LOGISTIC_REGR:
        # plot the confusion matrix
        cm = confusion_matrix(ACT, round(PRED,0))
        plt.imshow(cm, cmap=plt.cm.Blues)
        plt.colorbar()
        # add numbers to the confusion matrix
```

```
        for i in range(cm.shape[0]):
            for j in range(cm.shape[1]):
                plt.text(x=j, y=i, s=cm[i, j], va='center', ha='center')
        # xticks and yticks only 0-1
        plt.xticks([0,1], [0,1])
        plt.yticks([0,1], [0,1])
        plt.xlabel("Predicted")
        plt.ylabel("Actual")
        plt.tight_layout()
        plt.savefig("Confusion_Matrix.png")
        plt.close()

    return [_cor_, _mape_, _mamte_, _mae_, _rmse_, _alpha_, _beta_, _accuracy_
        , _FP_, _TP_, _FN_, _TN_]

def export_metrics(ytr, pred_tr, yte, pred_te, __method__, ttr, tte,
    LOGISTIC_REGR, path_):
    errTr = error_metrics(ytr, pred_tr, LOGISTIC_REGR)
    errTe = error_metrics(yte, pred_te, LOGISTIC_REGR)
    errTr.append(ttr)
    errTe.append(tte)
    df = pd.DataFrame([errTr, errTe])
    df.columns = ["R", "MAPE", "MAMPE", "MAE", "RMSE", "alpha", "beta", "Acc",
        "FP", "TP", "FN", "TN", "Sec"]
    df.index = ["Train", "Test"]
    pd.options.display.float_format = '{:,.2f}'.format
    df.to_excel(path_+__method__+"_error_metrics.xlsx", index=True)
    df[abs(df) > 1e10] = Inf
    print(df[["R", "MAPE", "MAMPE", "MAE", "RMSE", "Acc", "Sec"]])

def export_metrics_out(yout, pred_out, __method__, LOGISTIC_REGR):
    errOut = error_metrics(yout, pred_out, LOGISTIC_REGR)
    df = pd.DataFrame([errOut])
    df.columns = ["R", "MAPE", "MAMPE", "MAE", "RMSE", "alpha", "beta", "Acc",
        "FP", "TP", "FN", "TN"]
    df.index = ["Out_of_Sample_Metrics"]
    pd.options.display.float_format = '{:,.3f}'.format
    print(df)
    df.to_excel(__method__+"_error_metrics.xlsx", index=True)

def create_change_ml_dir(__method__):
    cwd = os.getcwd()
    if not os.path.exists("ML_Models"):
        os.makedirs("ML_Models")
    os.chdir("ML_Models")
    if not os.path.exists(__method__):
        os.makedirs(__method__)
    os.chdir(__method__)
    if not os.path.exists("Error_Analysis"):
        os.makedirs("Error_Analysis")
    if not os.path.exists("Sensitivity_Analysis"):
        os.makedirs("Sensitivity_Analysis")
    return cwd
```

```
def __pdf__(_x_, bins):
    rr = linspace(min(_x_)-1e-10, max(_x_)+1e-10, bins)
    ret = [len(where((rr[i-1] <= _x_) & (_x_ < rr[i]))[0]) for i in range(1,
        bins)]
    rr = (rr[1:] + rr[:-1])/2
    return divide(ret, sum(ret)), rr

def cdf_pdf_plot(varVals,var_,path_):
    bin = 20
    #bin = int(len(var1)/10)
    _pdf_,_range_ = __pdf__(varVals,bin)
    fig, ax = plt.subplots(ncols=1)
    ax.set_xlabel(var_)
    ax.set_ylabel("Frequency")
    ax.set_title("bins="+str(bin)+",␣samples="+str(len(varVals)))
    ax1 = ax.twinx()
    ax.bar(_range_,_pdf_,width=(_range_[1]-_range_[0])*0.9)

    # Add the number of occurrences on the vertical bars
    for i, v in enumerate(_pdf_):
        ax.text(_range_[i], v+0.00, str(int(_pdf_[i]*len(varVals))), ha='
            center', va='bottom')

    cdf = _pdf_.cumsum()
    cdf /= max(cdf)
    ax1.plot(_range_,cdf)
    ax1.set_ylabel("Cumulative␣Probability")
    ax1.grid(which='both')
    plt.savefig(path_+var_+"_PDF_CDF.png")
    plt.close()

def error_analysis(yy,pred,target_name,__method__,tag,path_err):
    err = yy - pred
    plt.scatter(yy, err)
    plt.xlabel(target_name)
    plt.ylabel(__method__.replace("_", "␣") + "␣Residual␣Errors␣"+tag)
    cor = pearsonr(err, yy)[0]
    plt.title("Pearson␣(Errors−"+target_name+")␣=␣" + "{:.5f}".format(cor) + "
        ,␣Average␣Error␣=␣" + "{:.5f}".format(mean(err)))
    rr = linspace(min(yy), max(yy), len(yy))
    plt.plot(rr, percentile(err, 5)*ones(len(rr)), color='black', label='5%␣
        Percentile', linestyle='—')
    plt.plot(rr, percentile(err, 50)*ones(len(rr)), color='black', label='
        Median', linestyle='−')
    plt.plot(rr, percentile(err, 95)*ones(len(rr)), color='black', label='95%␣
        Percentile', linestyle=':')
    plt.legend(loc='best')
    plt.savefig(path_err+__method__ + "_Errors_"+tag+".png")
    plt.close()

    cdf_pdf_plot(err,__method__ + "_Errors_"+tag,path_err)


def check_fix_multicolinearity(Xtr, Xte, features_names, ROOT_DIR):
```

```
    try:
        k = matrix_rank(Xtr)
        print("Rank of Xtr: ", k, " out of ", Xtr.shape[1], " features")
        if k<Xtr.shape[1]:
            q, r, p = linalg.qr(Xtr, pivoting=True)
            idx = p[:k]
            inds_exclude = delete(arange(Xtr.shape[1]), idx)
            istd = where(std(Xtr,axis=0)==0)[0]
            istd = np_append(istd, where(isnan(std(Xtr,axis=0)))[0])
            if len(istd)>0:
                print(len(istd),"features with zeros std:",features_names[istd
                    ])
            inds_exclude = np_append(inds_exclude, istd)
            inds_exclude = sort(unique(inds_exclude))
            print("Multicolinearity detected, excluding features: ", [i for i
                in features_names[inds_exclude]])
            Xtr = delete(Xtr, inds_exclude, axis=1)
            Xte = delete(Xte, inds_exclude, axis=1)
            features_names = delete(features_names, inds_exclude)
            with open(ROOT_DIR + "excluded_features.txt", "w") as f:
                for i in inds_exclude:
                    f.write(str(i) + "\n")
        else:
            print("No multicolinearity detected.")
    except Exception as ex1:
        print(ex1)
    return Xtr, Xte, features_names


def delete_missing_values_rows(XX, Y):
    try:
        missing_values = array([]).astype(int)
        for i in range(XX.shape[0]):
            for j in range(XX.shape[1]):
                if isnan(XX[i,j]) or isinf(XX[i,j]):
                    missing_values = np_append(missing_values, i)
            if isnan(Y[i]) or isinf(Y[i]):
                missing_values = np_append(missing_values, i)
        missing_values = unique(missing_values) # remove duplicates
        print("Number of missing values: ", len(missing_values),"::",
            missing_values)
        # remove rows with missing values
        XX = delete(XX, missing_values, axis=0)
        Y = delete(Y, missing_values, axis=0)
        # export_notebook_to_html()
    except Exception as ex1:
        print(ex1)

    return XX, Y

def compute_distances(XY):
    D=XY@XY.T
    g=diag(D)
    return -2*D+g+g.T
```

```
def delete_identical_rows(XX, Y):
    cwd = os.getcwd()
    try:
        XY = c_[XX, Y]
        i_del = array([])
        obs = XY.shape[0]

        D = compute_distances(XY)
        for i in range(obs):
            for j in range(i):
                if D[i,j] == 0:
                    i_del = np_append(i_del, i)
        D = 0
        print(len(i_del),"Identical rows found (XX, Y).")

        iok = []
        for i in range(obs):
            if i not in i_del:
                iok.append(i)
        print("NOF samples in new dataset =",len(iok),"in initial dataset =",
            obs)

        XX = XX[iok,:]
        Y = Y[iok]

    except Exception as ex1:
        print(ex1)
        os.chdir(cwd)
    return XX, Y

def read_out_of_sample_data(ROOT_DIR):
    s_all = []
    for file in os.listdir(ROOT_DIR+"Predict"):
        if file.endswith(".xlsx"):
            s_all.append(ROOT_DIR + "Predict" + os.sep + file)

    if len(s_all)>1:
        print("More than 1 files exists in the directory")
    elif len(s_all)==0:
        print("No files exist in the directory")
        return None
    else:
        my_file = s_all[0]
        print("
            ##############################################################################
            ")
        print(my_file)
        print("
            ##############################################################################
            ")
    df = pd.read_excel(my_file)
```

```
    cols = df.columns
    df = df.to_numpy()

    inds_exclude = []
    if os.path.isfile(ROOT_DIR + "excluded_features.txt"):
        with open(ROOT_DIR + "excluded_features.txt", 'r') as f:
            inds_exclude = [int(line.strip()) for line in f]
        print("Multicolinearity was detected, excluding features: ", [i for i
            in cols[inds_exclude]])

    # export_notebook_to_html()

    return  delete(df[:,:-1], inds_exclude, axis=1), df[:,-1], delete(cols
        [:-1], inds_exclude), cols[-1]



def split_tr_vl(obs, ___perc_cv___, nof_folds,PERMUTE_TRAIN_TEST):
    obs_fold = int(math.ceil(___perc_cv___*obs))
    obs_valid = obs - obs_fold

    i = 1
    if PERMUTE_TRAIN_TEST:
        obs_scanned = sorted(shuffle(range(obs), random_state=i)[:obs_valid])
    else:
        obs_scanned = sorted(range(obs)[:obs_valid])
    vl_inds = [obs_scanned]
    __restart__ = False
    while i < nof_folds:
        i += 1
        obs_to_add = []
        if obs - len(obs_scanned) <= obs_valid + obs % obs_valid:
            __restart__ = True
            obs_to_add = list(set(range(obs)) - set(obs_scanned))
        else:
            __restart__ = False
            ii = list(set(range(obs)) - set(obs_scanned))
            if PERMUTE_TRAIN_TEST:
                obs_to_add = sorted(shuffle(ii, random_state=i)[:obs_valid])
            else:
                obs_to_add = sorted(ii[:obs_valid])
            obs_scanned = sorted(obs_scanned + obs_to_add)
        vl_inds.append(obs_to_add)
        if __restart__ and i < nof_folds:
            obs_scanned = sorted(shuffle(range(obs), random_state=i)[:
                obs_valid])
            vl_inds.append(obs_scanned)
            i += 1
    print("Lengths Validation: ", [len(ir) for ir in vl_inds])

    intersect_all = zeros((len(vl_inds), len(vl_inds)))
    for i in range(len(vl_inds)):
        for j in range(i+1, len(vl_inds)):
            intersect_all[i,j] = len(set(vl_inds[i]).intersection(vl_inds[j]))
```

```
    print("intersect_all Validation:\n", intersect_all)


    tr_inds = []
    for i in range(nof_folds):
        i_tr_all = list(set(range(obs)) - set(vl_inds[i]))
        tr_inds.append(i_tr_all)
    print("Lengths Train: ", [len(ir) for ir in tr_inds])

    lvalid = []
    ltrain = []
    for i in range(nof_folds):
        lvalid += vl_inds[i]
        ltrain += tr_inds[i]

    print("Total Validation Samples=", len(lvalid), " Total Train Samples=",
        len(ltrain), " out of:", obs)
    print("Unique Validation Samples=", len(set(lvalid)), " Unique Train
        Samples=", len(set(ltrain)), " out of:", obs)

    return tr_inds, vl_inds



def export_notebook_to_html():
    notebook_name = '__nbml__.py'
    output_file_name = 'output.html'

    exporter = HTMLExporter()
    output_notebook = nbformat.read(
                        os.path.join(os.path.dirname(os.path.abspath(__file__)
                            ),
                                     notebook_name),
                                     as_version=4)

    output, resources = exporter.from_notebook_node(output_notebook)
    codecs.open(output_file_name, 'w', encoding='utf-8').write(output)


def do_sensitivity(Xtr, features_names, target_name, PREDICTOR, __method__,
    path_sens):
    vars = Xtr.shape[1]
    _med_ = median(Xtr, axis=0).reshape(1, vars)
    _75q_ = quantile(Xtr, 0.75, axis=0).reshape(1, vars)
    _25q_ = quantile(Xtr, 0.25, axis=0).reshape(1, vars)
    sens_med_all = zeros((Xtr.shape[0], vars))
    sens_75q_all = zeros((Xtr.shape[0], vars))
    sens_25q_all = zeros((Xtr.shape[0], vars))

    for i in range(vars):
        xx_tr_sens = repeat(_med_, Xtr.shape[0], axis=0)
        xx_tr_sens[:, i] = Xtr[:, i]
        sens_med_all[:, i] = PREDICTOR(xx_tr_sens)
```

```
            xx_tr_sens = repeat(_75q_, Xtr.shape[0], axis=0)
            xx_tr_sens[:, i] = Xtr[:, i]
            sens_75q_all[:, i] = PREDICTOR(xx_tr_sens)

            xx_tr_sens = repeat(_25q_, Xtr.shape[0], axis=0)
            xx_tr_sens[:, i] = Xtr[:, i]
            sens_25q_all[:, i] = PREDICTOR(xx_tr_sens)
            print(f"Preparing Sensitivity Curves for feature: {i+1} of {vars}.")

        df = pd.DataFrame(sens_med_all, columns=features_names)
        df.to_excel(path_sens+"sens_med_all_"+__method__+".xlsx", index=False)

        df = pd.DataFrame(sens_75q_all, columns=features_names)
        df.to_excel(path_sens+"sens_75q_all_"+__method__+".xlsx", index=False)

        df = pd.DataFrame(sens_25q_all, columns=features_names)
        df.to_excel(path_sens+"sens_25q_all_"+__method__+".xlsx", index=False)

        # scatter plot the feature values vs the sensitivity curves
        for i in range(vars):
            plt.plot(Xtr[:, i], sens_med_all[:, i], 'x', label='Median')
            plt.plot(Xtr[:, i], sens_75q_all[:, i], 'x', label='75%')
            plt.plot(Xtr[:, i], sens_25q_all[:, i], 'x', label='25%')
            plt.xlabel(features_names[i])
            plt.ylabel(target_name)
            plt.legend()
            plt.savefig(path_sens+f"sensitivity_curve_{features_names[i]}.png")
            plt.close()
            print(f"Plotting Sensitivity Curves for feature: {i+1} of {vars}.")


def gather_all_sensitivity_curves(Xtr, features_names, target_name, ROOT_DIR):
    try:
        sens_all = zeros(len(features_names))
        for i in range(len(features_names)):
            plt.figure(figsize=(10, 5))
            for root, dirs, files in os.walk(os.path.join(ROOT_DIR, "ML_Models
                ")):
                for dir in dirs:
                    for root1, dirs1, files1 in os.walk(os.path.join(ROOT_DIR,
                        "ML_Models", dir)):
                        for dir1 in dirs1:
                            if dir1.startswith("Sensitivity_Analysis"):
                                for root2, dirs2, files2 in os.walk(os.path.
                                    join(ROOT_DIR, "ML_Models", dir, dir1)):
                                    for file in files2:
                                        if file.startswith("sens_med_all"):
                                            print(f"Reading Sensitivity Curves
                                                for: {file}")
                                            df = pd.read_excel(os.path.join(
                                                ROOT_DIR, "ML_Models", dir,
                                                dir1, file))
                                            iso = argsort(Xtr[:, i])
                                            plt.plot(Xtr[:, i][iso], df[
```

```
                                                    features_names[i]][iso],
                                                    marker='x', label=file[13:])
                                                plt.xlabel(features_names[i])
                                                if file.find("XGBoost") != -1:
                                                    sens_all[i] = df[
                                                        features_names[i]].max()-
                                                        df[features_names[i]].min
                                                        ()
                plt.ylabel(target_name)
                plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left',
                    borderaxespad=0.)
                plt.tight_layout()
                plt.savefig(ROOT_DIR + "ML_Models" + os.sep + "ALL_Sensitivity" +
                    os.sep + f"sensitivity_curve_{features_names[i]}.png")
                plt.close()
                print(f"Plotting Sensitivity Curves for feature: {i+1} of {len(
                    features_names)}.")

            iso = argsort(sens_all)
            plt.barh(range(len(features_names)), sens_all[iso], align='center')
            plt.yticks(range(len(features_names)), array(features_names)[iso])
            for i, v in enumerate(sens_all[iso]):
                plt.text(v + 0.01, i + 0.25, str(round(v, 3)), color='blue',
                    fontweight='bold')
            plt.xlabel("Sensitivity")
            plt.ylabel("Features")
            plt.tight_layout()
            plt.savefig(ROOT_DIR + "ML_Models" + os.sep + "ALL_Sensitivity" + os.
                sep + "All_Sensitivity_XGBoost.png")
            plt.close()
        except Exception as ex1:
            print(ex1)

# fit an function to the data
def fit_func(x, a, b):
    # return a * log(x) + b
    return a * x + b

#  a function to count the number of samples of train target per bin
# and the mae of the test set per bin and plot results
def plot_mae_per_bin(ytr, yte, yte_pred, target_name, __method__, path_):
    # count the number of samples of train target per bin
    __nof_nins__ = 20
    min_y = min(min(ytr), min(yte))-1e-10
    max_y = max(max(ytr), max(yte))+1e-10
    bins = linspace(min_y, max_y, __nof_nins__)
    ytr_bins = digitize(ytr, bins)
    ytr_bins_count = zeros(len(bins))
    for i in range(1, len(bins) + 1):
        ytr_bins_count[i-1] = sum(ytr_bins == i)

    # calculate the mae of the test set per bin
    yte_bins = digitize(yte, bins)
    yte_mae = zeros(len(bins))
```

```
    for i in range(1, len(bins) + 1):
        if sum(yte_bins == i) > 0:
            yte_mae[i-1] = mean(abs(yte[yte_bins == i] - yte_pred[yte_bins ==
                i]))

    inz = ytr_bins_count > 0
    ytr_bins_count = ytr_bins_count[inz]
    yte_mae = yte_mae[inz]
    iso = argsort(ytr_bins_count)
    ytr_bins_count = ytr_bins_count[iso]
    yte_mae = yte_mae[iso]
    # scatter plot the number of samples of train target per bin vs the mae of
        the test set per bin
    #  plot only where the number of samples of train target per bin is
        greater than 0
    plt.plot(ytr_bins_count, yte_mae, 'x', color='blue')
    plt.xlabel("Number of samples of train target per bin (NSTB)")
    plt.ylabel("MAE of the test set per bin (MAET)")

    popt, pcov = curve_fit(fit_func, ytr_bins_count, yte_mae)
    pred_fit = fit_func(ytr_bins_count, *popt)
    corr = corrcoef(yte_mae, pred_fit)[0,1]

    xx = (ytr_bins_count[:-1] + ytr_bins_count[1:]) / 2
    xx = np_append(xx, xx[-1] + 2*(ytr_bins_count[-1]-xx[-1]))
    xx = np_append(xx[0] - 2*(xx[0]-ytr_bins_count[0]), xx)
    if xx[0]<=0:
        xx[0]=1
    pred_fit = fit_func(xx, *popt)
    # plt.plot(xx, pred_fit, 'o-', label="Curve Fit: MAET=%.3f * log(NSTB) +
        %.3f \nPearson Correlation (Fit-MAET): %.3f" % (popt[0], popt[1], corr
        ))
    plt.plot(xx, pred_fit, 'o-', label="Curve Fit: MAET=%.3f * NSTB + %.3f \
        nPearson Correlation (Fit-MAET): %.3f" % (popt[0], popt[1], corr))
    # add the number of samples of train target per bin to the plot
    for i, v in enumerate(ytr_bins_count):
        plt.text(v + 0.01, yte_mae[i] + 0.01, str(int(v))+(";%.3f"%yte_mae[i])
            , color='blue')
    plt.legend()
    plt.savefig(path_ + f"MAE_per_bin_{__method__}.png")
    plt.close()



def gather_all_ML_metrics(ROOT_DIR):
    for root, dirs, files in os.walk(ROOT_DIR + "ML_Models"):
        for dir in dirs:
            for root1, dirs1, files1 in os.walk(ROOT_DIR + "ML_Models" + os.
                sep + dir):
                for file in files1:
                    # find all xlsx files containing the word "error_metrics",
                        read them and concantenate them
                    if file.endswith(".xlsx") and "error_metrics" in file:
                        df = pd.read_excel(ROOT_DIR + "ML_Models" + os.sep +
```

```
                dir + os.sep + file)
            # rename Unnamed: 0 column to "Dataset"
            df.rename(columns={"Unnamed: 0": "Dataset"}, inplace=
                True)
            # add a first column with the name of the model
            df.insert(0, "Model", file.split("_")[0])
            if "df_all" in locals():
                df_all = pd.concat([df_all, df], ignore_index=True
                    )
            else:
                df_all = df

    # Select even and odd rows using iloc
    odd_df = df_all.iloc[::2]  # select tr rows
    even_df = df_all.iloc[1::2]  # select te rows
    # Concatenate the two DataFrames in the desired order
    df_all = pd.concat([odd_df, even_df])
    # save the concantinated dataframe to a xlsx file
    df_all.to_excel(ROOT_DIR + "ML_Models" + os.sep + "All_ML_metrics.xlsx",
        index=False)

    # bar plot of the MAE of the test set for all models
    df_all = df_all[df_all["Dataset"] == "Test"]
    df_all = df_all[["Model","MAE"]]
    # sort the dataframe by MAE
    df_all = df_all.sort_values(by="MAE")
    # plot the bar plot
    plt.bar(df_all["Model"], df_all["MAE"])
    plt.xticks(rotation=45)
    plt.ylabel("MAE")
    plt.tight_layout()
    plt.savefig(ROOT_DIR + "ML_Models" + os.sep + "MAE_test_set.png")
    plt.close()


def make_lags(X, Y, features_names, target_name, lags):
    XX = Y[:-lags].copy()
    XX_names = [target_name+"_LAG_"+str(lags)]
    for i in range(1,lags):
        XX = c_[XX,Y[i:-lags+i]]
        XX_names.append(target_name+"_LAG_"+str(lags-i))
    Y = Y[lags:]
    for j in range(X.shape[1]):
        XX = c_[XX, X[:,j][:-lags]]
        XX_names.append(features_names[j]+"_LAG_"+str(lags))
        for i in range(1,lags):
            XX = c_[XX, X[:,j][i:-lags+i]]
            XX_names.append(features_names[j]+"_LAG_"+str(lags-i))
        XX = c_[XX, X[:,j][lags:]]
        XX_names.append(features_names[j])
    return XX, Y, array(XX_names)
```

17 Appendix: Code for descriptive_statistics.py

```
from import_libraries import *
from misc_functions import *


def descriptive_statistics(Xtr, Xte, features_names, ROOT_DIR):
    try:
        mean1 = mean(Xtr, axis=0)
        median1 = median(Xtr, axis=0)
        std1 = std(Xtr, axis=0)
        min1 = Xtr.min(axis=0)
        max1 = Xtr.max(axis=0)
        skewness1 = skew(Xtr, axis=0)
        kurtosis1 = kurtosis(Xtr, axis=0)
        # df = pd.DataFrame({'mean':mean1, 'median':median1, 'std':std1, 'min
            ':min1, 'max':max1, 'skewness':skewness1, 'kurtosis':kurtosis1},
            index=features_names)
        # df.to_excel("descriptive_statistics_train.xlsx", index=True)
        # print("Descriptive statistics saved in Descriptive_Statistics/
            descriptive_statistics_train.xlsx")

        # test set descriptive statistics
        mean2 = mean(Xte, axis=0)
        median2 = median(Xte, axis=0)
        std2 = std(Xte, axis=0)
        min2 = Xte.min(axis=0)
        max2 = Xte.max(axis=0)
        skewness2 = skew(Xte, axis=0)
        kurtosis2 = kurtosis(Xte, axis=0)
        # df = pd.DataFrame({'mean':mean2, 'median':median2, 'std':std2, 'min
            ':min2, 'max':max2, 'skewness':skewness2, 'kurtosis':kurtosis2},
            index=features_names)
        # df.to_excel("descriptive_statistics_test.xlsx", index=True)
        # print("Descriptive statistics saved in Descriptive_Statistics/
            descriptive_statistics_test.xlsx")

        # deferrences between train and test
        # df = pd.DataFrame({'mean':mean1-mean2, 'median':median1-median2, '
            std':std1-std2, 'min':min1-min2, 'max':max1-max2, 'skewness':
            skewness1-skewness2, 'kurtosis':kurtosis1-kurtosis2}, index=
            features_names)
        # df.to_excel("descriptive_statistics_train_test_differences.xlsx",
            index=True)
        # print("Descriptive statistics saved in Descriptive_Statistics/
            descriptive_statistics_train_test_differences.xlsx")

        # train, test and differences in one exls file with 3 sheets
        file_exp = ROOT_DIR + "Descriptive_Statistics" + os.path.sep + "
            descriptive_statistics_train_test_and_differences.xlsx"
        writer = pd.ExcelWriter(file_exp, engine='openpyxl')
        df = pd.DataFrame({'mean':mean1, 'median':median1, 'std':std1, 'min':
            min1, 'max':max1, 'skewness':skewness1,
```

```
                        'kurtosis':kurtosis1}, index=features_names)
        df.to_excel(writer, sheet_name='train')
        df = pd.DataFrame({'mean':mean2, 'median':median2, 'std':std2, 'min':
            min2, 'max':max2, 'skewness':skewness2,
                        'kurtosis':kurtosis2}, index=features_names)
        df.to_excel(writer, sheet_name='test')
        df = pd.DataFrame({'mean':100*(mean1-mean2)/mean1, 'median':100*(
            median1-median2)/median1, 'std':100*(std1-std2)/std1,
                        'min':100*(min1-min2)/min1, 'max':100*(max1-max2)/
                            max1, 'skewness':100*(skewness1-skewness2)/
                            skewness1,
                        'kurtosis':100*(kurtosis1-kurtosis2)/kurtosis1},
                            index=features_names)
        df.to_excel(writer, sheet_name='%differences')
        writer.book.save(file_exp)
        writer.close()
        print("Descriptive statistics saved in Descriptive_Statistics/
            descriptive_statistics_train_test_and_differences.xlsx")


    except Exception as ex1:
        print(ex1)


def plot_pdf_cdf_all(Xtr, ytr, features_names, target_name, ROOT_DIR):
    try:
        for i in range(Xtr.shape[1]):
            cdf_pdf_plot(Xtr[:,i],features_names[i]+"_Train",ROOT_DIR+"
                Descriptive_Statistics"+os.path.sep+"PDF_CDF"+os.path.sep)
        cdf_pdf_plot(ytr,target_name+"_Train",ROOT_DIR+"Descriptive_Statistics
            "+os.path.sep+"PDF_CDF"+os.path.sep)
        print("PDF and CDF saved in Descriptive_Statistics/PDF_CDF")
    except Exception as ex1:
        print(ex1)

def plot_all_timeseries(XX, features_names, YY, target_name, plot_type,
    ROOT_DIR):
    try:
        ##############################
        window_size = 30
        ##############################
        for i in range(XX.shape[1]):
            rolling_correlation = zeros(len(YY))
            for j in range(window_size,len(YY)):
                rolling_correlation[j] = corrcoef(YY[j-window_size:j],XX[j-
                    window_size:j,i])[0,1]
            rolling_correlation[isnan(rolling_correlation)] = 0

            fig, (ax1, ax2) = plt.subplots(nrows=2, ncols=1, figsize=(50, 12))
            ax1.plot(rolling_correlation)
            ax1.set_xlabel('Time')
            ax1.set_ylabel('Rolling Correlation')
            ax1.grid(True)
            ax1.xaxis.set_major_locator(plt.MultipleLocator(100))
```

```
            ax2.plot(XX[:,i], color='blue')
            ax2.set_xlabel('Time')
            ax2.set_ylabel(features_names[i], color='blue')
            ax2.grid(True, which='both', axis='both')
            ax2.xaxis.set_major_locator(plt.MultipleLocator(100))
            ax3 = ax2.twinx()
            ax3.plot(YY, color='red')
            ax3.set_ylabel(target_name, color='red')
            fig.savefig(ROOT_DIR + "Descriptive_Statistics" + os.sep + "
                TimeSeries" + os.sep + plot_type + "_" + features_names[i] + "
                .png", bbox_inches='tight')
            plt.close()

            moving_average = []
            hor = 365
            for ii in range(len(XX[:,i]), hor, -1):
                moving_average.append(mean(XX[:,i][ii:ii-hor:-1]))
            plt.figure(figsize=(50, 12))
            plt.plot(XX[:,i][:len(XX[:,i])-hor])
            plt.title(plot_type + "_" + features_names[i])
            plt.plot(moving_average)
            plt.title(plot_type + "_" + features_names[i]+"_moving_average"+
                str(hor))
            plt.savefig(ROOT_DIR + "Descriptive_Statistics" + os.sep + "
                TimeSeries" + os.sep + plot_type + "_" + features_names[i]+"
                _moving_average"+str(hor)+".png", bbox_inches='tight')
            plt.close()

            print("Ploting time series for " + features_names[i])

        moving_average = []
        hor = 365
        for i in range(len(YY), hor, -1):
            moving_average.append(mean(YY[i:i-hor:-1]))
        plt.figure(figsize=(50, 12))
        plt.plot(YY[:len(YY)-hor])
        plt.title(plot_type + "_" + target_name)
        plt.plot(moving_average)
        plt.title(plot_type + "_" + target_name+"_moving_average"+str(hor))
        plt.savefig(ROOT_DIR + "Descriptive_Statistics" + os.sep + "TimeSeries
            " + os.sep + plot_type + "_" + target_name+"_moving_average"+str(
            hor)+".png", bbox_inches='tight')
        plt.close()

        print("Time series saved in Descriptive_Statistics/TimeSeries")
    except Exception as ex1:
        print(ex1)

def plot_all_by_all_correlation_matrix(Xtr, ytr, features_names, target_name,
    ROOT_DIR):
    try:
        corr = corrcoef(Xtr, ytr, rowvar=False)
        names_all = features_names.copy().tolist()
        names_all.append(target_name)
```

```
fig = plt.figure()
ax = fig.add_subplot(111)
cax = ax.matshow(corr, vmin=-1, vmax=1)
fig.colorbar(cax)
ticks = arange(0,len(names_all),1)
ax.set_xticks(ticks)
plt.xticks(rotation=90)
ax.set_yticks(ticks)
ax.set_xticklabels(names_all)
ax.set_yticklabels(names_all)
plt.savefig(ROOT_DIR + "Descriptive_Statistics"+ os.path.sep + "
    All_by_All_Correlation_Matrix.png", bbox_inches='tight')
plt.close()
# df = pd.DataFrame(c_[Xtr, ytr], columns = features_names.union([
    target_name]))
# sns.pairplot(df, diag_kind='kde')
# df=0
# plt.savefig(ROOT_DIR + "Descriptive_Statistics"+ os.path.sep + "
    All_by_All_Correlation_Matrix_Full.png", bbox_inches='tight')
# plt.close()
print("All_by_All_Correlation_Matrix_saved_in_Descriptive_Statistics/
    All_by_All_Correlation_Matrix.png")

# scatter plot of all features vs target in one plot
fig, ax = plt.subplots(Xtr.shape[1], 1, figsize=(7, 5*Xtr.shape[1]))
for i in range(Xtr.shape[1]):
    ax[i].scatter(Xtr[:,i], ytr)
    ax[i].set_xlabel(features_names[i])
    ax[i].set_ylabel(target_name)
    ax[i].set_title("Pearson_Correlation:_"+str(round(corr[i,-1],3)))
    # add trendline
    z = polyfit(Xtr[:,i], ytr, 1)
    p = poly1d(z)
    ax[i].plot(Xtr[:,i],p(Xtr[:,i]),"r--")

fig.tight_layout()
plt.savefig(ROOT_DIR + "Descriptive_Statistics"+ os.path.sep + "
    All_Features_vs_Target.png", bbox_inches='tight')
plt.close()
print("All_Features_vs_Target_saved_in_Descriptive_Statistics/
    All_Features_vs_Target.png")


# Generate Map
nof_obj = len(names_all)
similarity = copy(corr)
for i in range(nof_obj):
    for j in range(nof_obj):
        similarity[i,j] /= corr[i,i]+corr[j,j]-similarity[i,j]

func_evals = 2*nof_obj*500
lb=-10.0*ones((nof_obj,2))
ub=copy(-lb)
xa=lb+(ub-lb)*rand(nof_obj,2)
```

```
        opti_xa=copy(xa)
        iter_opti = []; all_opti = []
        opti_fu = -Inf
        inz = logical_and((0.0<similarity), (similarity<1))
        opti_D = 0
        for iter in range(func_evals):
            i=randint(0,high=nof_obj)
            j=randint(0,2)
            xa[i,j]=lb[i,j] + rand()*(ub[i,j]-lb[i,j])
            G = xa@transpose(xa)
            g = diag(G).reshape(-1,1)
            D = sqrt(-2*G+g+transpose(g))
            D /= D.max()

            fu = corrcoef(D[inz],-log(similarity[inz]))[0,1]
            if fu>opti_fu:
                opti_xa=copy(xa)
                opti_fu=copy(fu)
                opti_D = copy(D)
                iter_opti.append(iter)
                all_opti.append(opti_fu)
            else:
                xa=copy(opti_xa)
            if iter==func_evals-1:
                print(iter, "Map is ready, Optimal Objective: ", opti_fu)

        plt.scatter(iter_opti, all_opti)
        plt.savefig(ROOT_DIR + "Descriptive_Statistics"+ os.path.sep + "
            Convergence History.png")
        plt.close()
        plt.scatter(opti_D[inz],-log(similarity[inz]))
        plt.savefig(ROOT_DIR + "Descriptive_Statistics"+ os.path.sep + "
            Distances vs Similarity.png")
        plt.close()

        corry = corr[:,-1]
        ss = [int(100*abs(corry[i])) for i in range(len(corry))]
        fig, ax = plt.subplots()
        plt.scatter(opti_xa[:,0],opti_xa[:,1],s=ss)
        texts = [plt.text(opti_xa[i,0], opti_xa[i,1], names_all[i],
                fontsize=10) for i in range(nof_obj)]
        plt.axis('off')
        adjust_text(texts, arrowprops=dict(arrowstyle='->', color='red'))
        # , arrowprops=dict(arrowstyle='->', color='red')
        plt.savefig(ROOT_DIR + "Descriptive_Statistics"+ os.path.sep + "map.
            png")
        plt.close()
    except Exception as ex1:
        print(ex1)

    # export_notebook_to_html()

def export_descriptive_per_bin(Xtr,Xte,ytr,yte,features_names,target_name,
    ROOT_DIR):
```

```
file_exp = ROOT_DIR + "Descriptive_Statistics" + os.path.sep + "
    descriptive_statistics_per_bin.xlsx"
writer = pd.ExcelWriter(file_exp, engine='openpyxl')

percentiles_ytr = []
percentiles_yte = []
qstart = [0,0, 0, 0, 50, 75, 95, 99]
qend =   [1,5,25,50,100,100,100,100]
for i in range(len(qstart)):
    percentiles_ytr.append((percentile(ytr,qstart[i]), percentile(ytr,qend
        [i])))
    percentiles_yte.append((percentile(yte,qstart[i]), percentile(yte,qend
        [i])))


significant_names = []
significant_perc = []
str_min_max = []
for i in range(len(percentiles_ytr)):
    ii = where((percentiles_ytr[i][0]<=ytr) & (ytr<=percentiles_ytr[i][1])
        )[0]
    str_= '{:.5e}'.format(percentiles_ytr[i][0])+"<=ytr<="+'{:.5e}'.format
        (percentiles_ytr[i][1])+"|"+str(len(ii))+"_ytr_values"
    DX = (Xtr[ii,:].max(axis=0) - Xtr[ii,:].min(axis=0))/(Xtr.max(axis=0)
        - Xtr.min(axis=0))
    iso = argsort(DX)
    str_min_max.append(features_names[iso[0]] + "_in_" + '{:.2e}'.format(
        Xtr[ii,iso[0]].min(axis=0))+"~"+'{:.2e}'.format(Xtr[ii,iso[0]].max
        (axis=0)))
    mean1 = mean(Xtr[ii,:], axis=0)[iso]
    median1 = median(Xtr[ii,:], axis=0)[iso]
    std1 = std(Xtr[ii,:], axis=0)[iso]
    min1 = Xtr[ii,:].min(axis=0)[iso]
    max1 = Xtr[ii,:].max(axis=0)[iso]
    skewness1 = skew(Xtr[ii,:], axis=0)[iso]
    kurtosis1 = kurtosis(Xtr[ii,:], axis=0)[iso]
    df1 = pd.DataFrame({'Dataset':"Train", 'mean':mean1, 'median':median1,
         'std':std1, 'min':min1, 'max':max1, 'skewness':skewness1,
                        'kurtosis':kurtosis1}, index=features_names[iso])
    significant_names.append(features_names[iso[0]])
    significant_perc.append(DX[iso[0]])

    ii = where((percentiles_yte[i][0]<=yte) & (yte<=percentiles_yte[i][1])
        )[0]
    str_ += "|"+str(len(ii))+"_yte_values"
    mean2 = mean(Xte[ii,:], axis=0)[iso]
    median2 = median(Xte[ii,:], axis=0)[iso]
    std2 = std(Xte[ii,:], axis=0)[iso]
    min2 = Xte[ii,:].min(axis=0)[iso]
    max2 = Xte[ii,:].max(axis=0)[iso]
    skewness2 = skew(Xte[ii,:], axis=0)[iso]
    kurtosis2 = kurtosis(Xte[ii,:], axis=0)[iso]
    df2 = pd.DataFrame({'Dataset':"Test", 'mean':mean2, 'median':median2,
```

```
            'std':std2, 'min':min2, 'max':max2, 'skewness':skewness2,
                                'kurtosis':kurtosis2}, index=features_names[iso])
        new_df = pd.concat([df1, df2])
        new_row = pd.DataFrame({'Dataset':[None], 'mean':[None], 'median':[
            None], 'std':[None], 'min':[None], 'max':[None], 'skewness':[None
            ],
                                'kurtosis':[None]}, index=[str_])
        new_df = pd.concat([new_df, new_row])
        new_df.to_excel(writer, sheet_name="q"+str(qstart[i])+"-q"+str(qend[i
            ]))

    writer.book.save(file_exp)
    writer.close()


    plt.barh(range(len(significant_names)), significant_perc, align='center')
    plt.yticks(range(len(significant_names)), str_min_max)
    for i, v in enumerate(significant_perc):
        plt.text(v + 0.01, i + 0.25, target_name+" in Q"+str(qstart[i])+"-Q"+
            str(qend[i]), color='blue', fontweight='bold')
    plt.xlabel("Percentage of Xmax-Xmin in Quantile")
    plt.tight_layout()
    # Get the current axis object
    ax = plt.gca()
    # Remove the top and right spines of the axis
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    plt.savefig(ROOT_DIR + "Descriptive_Statistics"+ os.path.sep + "
        Percentage_of_Xmax-Xmin_in_Quantile.png")
    plt.close()


def plot_short_tree(Xtr, ytr, features_names, target_name, ROOT_DIR):

    leaf_nodes_all = []
    unique_leaf_nodes_all = []
    for __DEPTH__ in range(1,4):
        print("Computing short tree, with depth:",__DEPTH__)
        # create decision tree classifier with max_depth=3
        dtree = DecisionTreeRegressor(max_depth=__DEPTH__, criterion='
            absolute_error', min_samples_leaf = 20, random_state=0)

        # fit the model with iris data
        dtree.fit(Xtr, ytr)

        # print out the tree structure, thresholds, and leaves
        tree_rules = export_text(dtree, feature_names=list(features_names))
        print(tree_rules)

        # create subplots grid
        # fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(20,5))
        fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(21,10), gridspec_kw={
            'width_ratios': [2, 1]})
        # plot decision tree on left-hand side
```

```
    plot_tree(dtree, feature_names=list(features_names), filled=True, ax=
        ax1, node_ids = False, proportion= True)
    # plot PDF and CDF of y_train on right-hand side
    sns.kdeplot(ytr, ax=ax2, fill=True, color='r', label="PDF-ALL")
    # sns.kdeplot(ytr, ax=ax2, cumulative=True, color='r', label="CDF-ALL
        ")
    ax2.set_ylabel('PDF for all samples and for each leaf node')
    ax2.set_xlabel(target_name)
    ax2.grid()

    ################################################
    leaf_nodes = dtree.apply(Xtr)
    unique_leaf_nodes = unique(leaf_nodes)
    leaf_nodes_all.append(leaf_nodes)
    unique_leaf_nodes_all.append(unique_leaf_nodes)
    for i in range(len(unique_leaf_nodes)):
        node = unique_leaf_nodes[i]
        indices = where(leaf_nodes == node)[0]
        sns.kdeplot(ytr[indices], ax=ax2, fill=False, label="LEAF-"+str(i
            +1))
    ################################################
    plt.tight_layout()
    plt.legend()
    # save figure and close plot object
    plt.savefig(ROOT_DIR+"Descriptive_Statistics"+os.sep+"Tree"+os.sep +
        target_name+'_tree_with_pdf_for_all_leaves_depth_'+str(__DEPTH__)+
        '.png')
    plt.close()

    feature_importance = dtree.tree_.compute_feature_importances()
    iso = argsort(feature_importance)[::-1]
    iPos = where(feature_importance[iso]>0.001)[0]
    plt.barh(range(len(iPos)),feature_importance[iso][iPos])
    plt.yticks(range(len(iPos)), features_names[iso][iPos])
    plt.tight_layout()
    plt.savefig(ROOT_DIR+"Descriptive_Statistics"+os.sep+"Tree"+os.sep +
        target_name+'_feature_importance_depth_'+str(__DEPTH__)+'.png')
    plt.close()

cdf_pdf_plot(ytr,target_name+"_ALL",ROOT_DIR+"Descriptive_Statistics"+os.
    sep+"Tree"+os.sep)
for j in range(len(leaf_nodes_all)):
    leaf_nodes = leaf_nodes_all[j]
    unique_leaf_nodes = unique_leaf_nodes_all[j]
    for i in range(len(unique_leaf_nodes)):
        node = unique_leaf_nodes[i]
        indices = where(leaf_nodes == node)[0]
        cdf_pdf_plot(ytr[indices],
                     "Level-"+str(j+1)+"_LEAF-"+str(i+1),
                     ROOT_DIR+"Descriptive_Statistics"+os.sep+"Tree"+os.
                         sep)
```

18 Appendix: Train Linear Regression - ml_linear_regression.py

```
from import_libraries import *
from misc_functions import *



def plot_pvalues(p_vals, features_names, path_):
    iso = argsort(p_vals)
    scf = int(len(features_names)/10)+1
    plt.figure(figsize=(10*scf, 10))
    plt.rcParams.update({'font.size': 7*scf})
    tik = arange(len(p_vals))/7
    plt.bar(tik, p_vals[iso], width=0.05)
    plt.xticks(tik, features_names, rotation=45, ha='right')
    plt.ylabel("p-values")
    plt.tight_layout()
    plt.savefig(path_ + "p_Values.png")
    plt.close()
    plt.rcParams.update({'font.size': 11})

def plot_normalised_weights(Xtr, ytr, features_names, LOGISTIC_REGR, path_):

    Xtr_norm = (Xtr - Xtr.min(axis=0)) / (Xtr.max(axis=0) - Xtr.min(axis=0))
    ytr_norm = (ytr - ytr.min(axis=0)) / (ytr.max(axis=0) - ytr.min(axis=0))

    if LOGISTIC_REGR:
        res = regression_bak(Xtr_norm, logit((ytr_norm*0.98)+0.01))
    else:
        res = regression_bak(Xtr_norm, ytr_norm)

    aa = res.aa

    iso = argsort(abs(aa))[::-1]
    scf = int(len(features_names)/10)+1
    plt.figure(figsize=(10*scf, 10))
    plt.rcParams.update({'font.size': 7*scf})
    tik = arange(len(aa))/7
    plt.bar(tik, aa[iso], width=0.05)
    plt.xticks(tik, features_names, rotation=45, ha='right')
    plt.title("Normalised Regression Weights")
    plt.xlabel("Features")
    plt.ylabel("Weights")
    plt.tight_layout()
    plt.savefig(path_ + "Normalised_Regression_Weights.png")
    plt.close()
    plt.rcParams.update({'font.size': 11})

def export_equation_in_excel(ww, LOGISTIC_REGR, path_, ROOT_DIR):
    col_excel = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "
       M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]
    col_excel0 = list(col_excel)
    for i in range(len(col_excel0)):
```

```
        col_excel = col_excel + [col_excel0[i]+c for c in col_excel0]
    col_excel = array(col_excel)

    inds_exclude = []
    if os.path.isfile(ROOT_DIR + "excluded_features.txt"):
        with open(ROOT_DIR + "excluded_features.txt", 'r') as f:
            inds_exclude = [int(line.strip()) for line in f]
        inds_exclude = array(inds_exclude)
        print("Multicolinearity was detected, excluding columns from excel: ",
            [i for i in col_excel[inds_exclude]])
    col_excel = delete(col_excel, inds_exclude, axis=0)

    equation_excel = "="
    if LOGISTIC_REGR:
        equation_excel += "expit("
    for i in range(len(ww)):
        str_ = "{:.15E}".format(ww[i])
        if ww[i] >= 0:
            str_ = "+" + str_
        equation_excel += "{}*{}2".format(str_, col_excel[i])
    if LOGISTIC_REGR:
        equation_excel += ")"

    # write equation in txt file
    with open(path_ + "Equation.txt", "w") as file:
        file.write(equation_excel)


def do_regression(Xtr, Xte, ytr, yte, features_names, target_name, ROOT_DIR,
    LOGISTIC_REGR):
    global pred_tr, pred_te, res, do_logistic
    do_logistic = LOGISTIC_REGR
    __method__ = "LinRegr"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_err = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"Error_Analysis"+
        os.sep
    path_sens = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
        Sensitivity_Analysis"+os.sep
    try:
        t0=time()
        if LOGISTIC_REGR:
            res = regression_bak(Xtr, logit((ytr*0.98)+0.01))
        else:
            res = regression_bak(Xtr, ytr)
        ttr = time()-t0
        with open(path_ + "LinRegrResults.csv", "w") as file:
            for field in res._fields:
                vector = getattr(res, field)
                file.write(field + ',')
                # write each element of the vector on a separate column
                # if is string
                if type(vector) == str:
                    file.write(vector + ',')
                else:
```

```
                    # if is vector
                    if isinstance(vector, ndarray):
                        for element in vector:
                            file.write(str(element) + ',')
                    # if is scalar
                    else:
                        file.write(str(vector) + ',')
                file.write('\n')

        pred_tr = predict_lin_regr(Xtr)
        t0=time()
        for i in range(10):
            pred_te = predict_lin_regr(Xte)
        tte=(time()-t0)/10

        plot_pvalues(res.p_vals, features_names, path_)

        plot_normalised_weights(Xtr, ytr, features_names, LOGISTIC_REGR, path_
            )

        export_equation_in_excel(res.aa, LOGISTIC_REGR, path_, ROOT_DIR)

        do_sensitivity(Xtr, features_names, target_name, predict_lin_regr,
            __method__, path_sens)

        plot_mae_per_bin(ytr, yte, pred_te, target_name, __method__, path_)

        error_analysis(ytr,pred_tr,target_name,__method__,"Train",path_err)
        error_analysis(yte,pred_te,target_name,__method__,"Test",path_err)

        plot_target_vs_predicted(ytr, pred_tr, target_name, __method__, "Train
            ",path_)
        plot_target_vs_predicted(yte, pred_te, target_name, __method__, "Test"
            ,path_)
        export_metrics(ytr, pred_tr, yte, pred_te, __method__, ttr, tte,
            LOGISTIC_REGR, path_)

        print("See results in folder:", path_)

        # export_notebook_to_html()
        gather_all_ML_metrics(ROOT_DIR)

    except Exception as ex1:
        print(ex1)


def predict_lin_regr(Xtr):
    global res, do_logistic
    if do_logistic:
        return expit(dot(Xtr, res.aa))
    else:
        return dot(Xtr, res.aa)
```

```
def predict_linregr(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR):
    global res, cwd
    __method__ = "LinRegr"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_pred = ROOT_DIR+"Predict"+os.sep+__method__+os.sep
    # try to load the results from the CSV file
    try:
        with open(path_ + "LinRegrResults.csv", 'r') as f:
            reader = csv.reader(f)
            row = next(reader)

        for i in range(1, len(row)-1):
            row[i] = float(row[i])
        aa = array(row[1:-1])

        if LOGISTIC_REGR:
            pred_out = expit(dot(Xout, aa))
        else:
            pred_out = dot(Xout, aa)
    except Exception as e:
        print("Error: ", e)
        return

    # save predictions to file
    with open(path_pred+"Predictions_"+__method__+".csv", "w") as file:
        for yi in pred_out:
            file.write(str(yi) + '\n')

    plot_target_vs_predicted(yout, pred_out, target_name, __method__, "Out",
        path_pred)
    export_metrics_out(yout, pred_out, path_pred + __method__ + "_Out",
        LOGISTIC_REGR)
    error_analysis(yout, pred_out, target_name, __method__, "Out", path_pred)

    print("See results in folder: ", path_pred)
```

19 Appendix: Train Polynomial Regression - ml_nlregr.py

```
from import_libraries import *
from misc_functions import *


def export_equation_txt(opti_aa, feat_names, poly_degree, target_name,
    combins_list, best_ira, path_):
    __formula__ = ""
    for i in range(len(opti_aa)):
        str_val = "{:.5E}".format(opti_aa[i])
        if opti_aa[i] > 0:
            str_val = "+" + str_val
        __formula__ = __formula__ + str_val
        for j in range(poly_degree):
            __formula__ = __formula__ + "*" + feat_names[combins_list[best_ira
                [i], j]]
    __formula__ = target_name + "=" + __formula__
    print(__formula__)
    with open(path_ + "__formula__.txt", "w") as f:
        f.write(__formula__)


def export_equation_in_excel(opti_aa, LOGISTIC_REGR, poly_degree, best_ira,
    vars, path_, ROOT_DIR):
    equation_excel = ""
    col_excel = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "
       M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]
    col_excel += [f"A{col}" for col in col_excel]
    col_excel = array(col_excel)

    inds_exclude = []
    if os.path.isfile(ROOT_DIR + "excluded_features.txt"):
        with open(ROOT_DIR + "excluded_features.txt", 'r') as f:
            inds_exclude = [int(line.strip()) for line in f]
        inds_exclude = array(inds_exclude)
        print("Multicolinearity was detected, excluding columns from excel: ",
            [i for i in col_excel[inds_exclude]])
    col_excel = delete(col_excel, inds_exclude, axis=0)

    for i in range(len(opti_aa)):
        str_val = "{:.15E}".format(opti_aa[i])
        if opti_aa[i] >= 0:
            str_val = "+" + str_val
        equation_excel = equation_excel + str_val
        for j in range(poly_degree):
            ij = combins_list[best_ira[i], j]
            if ij == 0:
                str_ij = "1"
            else:
                str_ij = col_excel[ij - 1] + "2"
            equation_excel = equation_excel + "*" + str_ij
```

```
    equation_excel = "=" + equation_excel
    print(equation_excel)
    with open(path_ + "__equation_excel__.txt", "w") as f:
        f.write(equation_excel)


def nl_features_mat(XX, INDS, NOFNL):
    XXX = XX[:, INDS[:, 0]].copy()
    for i in range(1, NOFNL):
        XXX *= XX[:, INDS[:, i]]
    return XXX


def do_nlregr(Xtr, Xte, ytr, yte, features_names, target_name, LOGISTIC_REGR,
    PERMUTE_TRAIN_TEST, ROOT_DIR):
    global pred_tr, pred_te, combins_list, best_ira, poly_degree, aa,
        do_logistic

    do_logistic = LOGISTIC_REGR

    __method__ = "NLRegr"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_err = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"Error_Analysis"+
        os.sep
    path_sens = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
        Sensitivity_Analysis"+os.sep

    try:
        t0=time()

        # split train to train and validation
        ___perc_cv___ = 0.8; nof_folds = 5; obs = len(ytr)
        tr_inds, vl_inds = split_tr_vl(obs, ___perc_cv___, nof_folds,
            PERMUTE_TRAIN_TEST)

        Xtr = c_[ones(Xtr.shape[0]), Xtr]
        Xte = c_[ones(Xte.shape[0]), Xte]
        feat_names = features_names.insert(0, '1')

        max_n_rounds = 1_000
        poly_degree = 3
        vars = Xtr.shape[1]
        combins_list = array(list(combinations_with_replacement(range(vars),
            poly_degree)))
        n_comb = combins_list.shape[0]
        print("Total Combinations:",n_comb,", for",vars,"variables and
            polynomial degree",poly_degree,".")

        ira = array(arange(vars))
        err_vl_all1 = []
        err_tr_all1 = []
        pred_te = zeros(len(yte))
        MNL_te = nl_features_mat(Xte, combins_list[ira,:], poly_degree)
        for j in range(nof_folds):
```

```
    if LOGISTIC_REGR:
        aa = lstsq(nl_features_mat(Xtr[tr_inds[j], :], combins_list[
            ira, :], poly_degree),
                logit((ytr[tr_inds[j]]*0.98)+0.01), rcond=None)[0]
    else:
        aa = lstsq(nl_features_mat(Xtr[tr_inds[j], :], combins_list[
            ira, :], poly_degree),
                ytr[tr_inds[j]], rcond=None)[0]
    pred_vl = nl_features_mat(Xtr[vl_inds[j], :], combins_list[ira,
        :], poly_degree) @ aa
    err_vl_all1.append(corrcoef(pred_vl, ytr[vl_inds[j]])[0,1])
    pred_tr = nl_features_mat(Xtr[tr_inds[j], :], combins_list[ira,
        :], poly_degree) @ aa
    err_tr_all1.append(corrcoef(pred_tr, ytr[tr_inds[j]])[0,1])
    pred_te += MNL_te @ aa
pred_te /= nof_folds
err_te = corrcoef(pred_te, yte)[1,0]
err_vl_all1 = array(err_vl_all1)
err_vl_all1[isnan(err_vl_all1)] = 0
err_tr_all1 = array(err_tr_all1)
err_tr_all1[isnan(err_tr_all1)] = 0
err_vl = quantile(err_vl_all1, 0.05)
err_tr = quantile(err_tr_all1, 0.05)
best_acc = err_vl.copy()
if isnan(best_acc):
    best_acc = 0
print("Starting Accuracy = ", best_acc)
best_ira = ira.copy()

acc_tr_all = [err_tr]
acc_vl_all = [err_vl]
acc_te_all = [err_te]
i_all = [0]

for iter in range(max_n_rounds):
    try:
        rr = rand()
        if rr < 0.35:
            ira[randint(0,len(ira))] = choice([x for x in range(n_comb
                ) if x not in ira])
        elif rr < 0.7:
            ira = np_append(ira, choice([x for x in range(n_comb) if x
                 not in ira]))
        else:
            # delete a random index of ira
            ira = np_delete(ira, randint(0,len(ira)))

        err_vl_all1 = []
        err_tr_all1 = []
        pred_te = zeros(len(yte))
        MNL_te = nl_features_mat(Xte, combins_list[ira,:], poly_degree
            )
        for j in range(nof_folds):
            if LOGISTIC_REGR:
```

```
                aa = lstsq(nl_features_mat(Xtr[tr_inds[j], :],
                    combins_list[ira, :], poly_degree),
                        logit((ytr[tr_inds[j]]*0.98)+0.01), rcond=None
                            )[0]
            else:
                aa = lstsq(nl_features_mat(Xtr[tr_inds[j], :],
                    combins_list[ira, :], poly_degree),
                        ytr[tr_inds[j]], rcond=None)[0]
            pred_vl = nl_features_mat(Xtr[vl_inds[j], :], combins_list
                [ira, :], poly_degree) @ aa
            err_vl_all1.append(corrcoef(pred_vl, ytr[vl_inds[j]])
                [0,1])
            pred_tr = nl_features_mat(Xtr[tr_inds[j], :], combins_list
                [ira, :], poly_degree) @ aa
            err_tr_all1.append(corrcoef(pred_tr, ytr[tr_inds[j]])
                [0,1])
            pred_te += MNL_te @ aa
        pred_te /= nof_folds

        err_te = corrcoef(pred_te, yte)[1,0]
        err_vl_all1 = array(err_vl_all1)
        err_vl_all1[isnan(err_vl_all1)] = 0
        err_tr_all1 = array(err_tr_all1)
        err_tr_all1[isnan(err_tr_all1)] = 0
        err_vl = quantile(err_vl_all1, 0.05)
        err_tr = quantile(err_tr_all1, 0.05)

        if err_vl > best_acc:
            best_acc = err_vl.copy()
            best_ira = ira.copy()
            print(time()-t0,iter, err_tr, best_acc, err_te, len(
                best_ira))
            acc_tr_all.append(err_tr)
            acc_vl_all.append(err_vl)
            acc_te_all.append(err_te)
            i_all.append(iter+1)
        else:
            ira = best_ira.copy()
    except KeyboardInterrupt:
        print('KeyboardInterrupt: Stopped by user')
        break

plt.plot(i_all, acc_tr_all, label='train')
plt.plot(i_all, acc_vl_all, label='validation')
plt.plot(i_all, acc_te_all, label='test')
plt.legend()
plt.savefig(path_ + "acc.png")
plt.close()

ipl = int(len(i_all)/2)
plt.plot(i_all[ipl:], acc_tr_all[ipl:], label='train')
plt.plot(i_all[ipl:], acc_vl_all[ipl:], label='validation')
plt.plot(i_all[ipl:], acc_te_all[ipl:], label='test')
plt.legend()
```

```
        plt.savefig(path_ + "acc50perc.png")
        plt.close()


        aa = lstsq(nl_features_mat(Xtr, combins_list[best_ira, :], poly_degree
            ), ytr, rcond=None)[0]
        pred_tr = nl_features_mat(Xtr, combins_list[best_ira, :], poly_degree)
             @ aa
        ttr = time()-t0
        t0=time()
        for i in range(10):
            pred_te = nl_features_mat(Xte, combins_list[best_ira,:],
                poly_degree) @ aa
        tte=(time()-t0)/10

        Xtr = Xtr[:,1:]
        Xte = Xte[:,1:]

        export_equation_txt(aa, feat_names, poly_degree, target_name,
            combins_list, best_ira, path_)
        export_equation_in_excel(aa, LOGISTIC_REGR, poly_degree, best_ira, Xtr
            .shape[1], path_, ROOT_DIR)

        do_sensitivity(Xtr, features_names, target_name, predict_nlregr,
            __method__, path_sens)

        plot_mae_per_bin(ytr, yte, pred_te, target_name, __method__, path_)

        error_analysis(ytr,pred_tr,target_name,__method__,"Train",path_err)
        error_analysis(yte,pred_te,target_name,__method__,"Test",path_err)

        plot_target_vs_predicted(ytr, pred_tr, target_name, __method__, "Train
            ",path_)
        plot_target_vs_predicted(yte, pred_te, target_name, __method__, "Test"
            ,path_)
        export_metrics(ytr, pred_tr, yte, pred_te, __method__, ttr, tte,
            LOGISTIC_REGR, path_)

        print("See_results_in_folder:", path_)

        # export_notebook_to_html()
        gather_all_ML_metrics(ROOT_DIR)

    except Exception as ex1:
        print(ex1)


def predict_nlregr(XX):
    global combins_list, best_ira, poly_degree, aa, do_logistic

    pred = nl_features_mat(c_[ones(XX.shape[0]), XX], combins_list[best_ira,
        :], poly_degree) @ aa

    if do_logistic:
```

```
    return expit(pred)
else:
    return pred
```

20 Appendix: Train XGBoost - ml_xgboost.py

```
# Description: This file contains the code for training and evaluating XGBoost
    models.


from import_libraries import *
from misc_functions import *


def do_xgboost(Xtr,Xte,ytr,yte,features_names,target_name,__thres_early_stop__
    ,__thres_min_tune_rounds__,PERMUTE_TRAIN_TEST,LOGISTIC_REGR,ROOT_DIR):
    try:
        t0=time()
        max_depth = list(arange(1, 11))
        learning_rate = list(np_round(arange(0.01, 0.51, 0.02), 2))
        n_estimators = list(concatenate((arange(1,11),arange(20, 101, 10),
            arange(200, 1001, 100))))
        colsample_bytree = list(np_round(arange(0.25, 1.01, 0.05), 2))
        subsample = list(np_round(arange(0.25, 1.01, 0.05), 2))
        combinations = list(product(max_depth, learning_rate, n_estimators,
            colsample_bytree, subsample))
        # randomly permute the combinations
        combinations = shuffle(combinations, random_state=0)

        # split train to train and validation
        ___perc_cv___ = 0.8; nof_folds = 5; obs = len(ytr)
        tr_inds, vl_inds = split_tr_vl(obs,___perc_cv___,nof_folds,
            PERMUTE_TRAIN_TEST)

        # train xgboost on each combination and evaluate on validation set
        acc_tr_all = array([]); acc_vl_all = array([]); acc_te_all = array([])
        opti_tr = -1; opti_vl = -1; opti_te= -1
        best_combination = None
        for i, (max_depth, learning_rate, n_estimators, colsample_bytree,
            subsample) in enumerate(combinations):
            # catch exception if the combination is not valid
            try:
                print("Training_XGBoost_Model",i+1,"/>",
                    __thres_min_tune_rounds__,"/",len(combinations),":",
                                        (max_depth, learning_rate,
                                            n_estimators, colsample_bytree
                                            , subsample))
                xgboost = xgb.XGBRegressor(max_depth=max_depth, learning_rate=
                    learning_rate, n_estimators=n_estimators,
                                        colsample_bytree=colsample_bytree,
                                            subsample=subsample, objective='
                                            reg:squarederror')
                acc_tr = 0; acc_vl = 0; acc_te = 0
                for fold in range(nof_folds):
                    xgboost.fit(Xtr[tr_inds[fold],:], ytr[tr_inds[fold]])
                    pred_tr = xgboost.predict(Xtr[tr_inds[fold],:])
                    pred_vl = xgboost.predict(Xtr[vl_inds[fold],:])
```

```
            pred_te = xgboost.predict(Xte)
            acc_tr += pearsonr(ytr[tr_inds[fold]], pred_tr).
                correlation
            acc_vl += pearsonr(ytr[vl_inds[fold]], pred_vl).
                correlation
            acc_te += pearsonr(yte, pred_te).correlation
        acc_tr /= nof_folds; acc_vl /= nof_folds; acc_te /= nof_folds
        if isnan(acc_tr):
            acc_tr = 0
        if isnan(acc_vl):
            acc_vl = 0
        if isnan(acc_te):
            acc_te = 0
        acc_tr_all = np_append(acc_tr_all, acc_tr)
        acc_vl_all = np_append(acc_vl_all, acc_vl)
        acc_te_all = np_append(acc_te_all, acc_te)
        # print all the r2 scores
        vl_75 = percentile(acc_vl_all, 75)
        vl_max = max(acc_vl_all)
        slope = (vl_max - vl_75)/(0.25*(i+1))
        print(datetime.datetime.now().strftime("%H:%M:%S"),"Comb:",i,"
            R2 Score:: Train:",acc_tr,"Validation:",acc_vl,"Test:",
            acc_te,
            "\nMax Val:",vl_max,"Q75 Val:",vl_75,"Slope:",slope)
        imax = argmax(acc_vl_all)
        best_combination = combinations[imax]
        print(best_combination)
        if i>__thres_min_tune_rounds__ and slope <__thres_early_stop__
            :
            print("Early Stopping, vl_max:",vl_max,"vl_95:",vl_75)
            break
    except:
        break

# find minimum length among acc_tr_all, acc_vl_all, acc_te_all, in
    case of exception
max_len = min([len(acc_tr_all), len(acc_vl_all), len(acc_te_all)])
# make all the arrays of same length
acc_tr_all = acc_tr_all[:max_len]
acc_vl_all = acc_vl_all[:max_len]
acc_te_all = acc_te_all[:max_len]

imax = argmax(acc_vl_all)
best_combination = combinations[imax]
print("Best Combination:",best_combination)

ttr = time()-t0
__method__ = "XGBoost"
path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
path_err = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
    Error_Analysis"+os.sep
path_sens = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
    Sensitivity_Analysis"+os.sep
```

```
with open(path_ + "Best_Combination.txt", "w") as f:
    f.write(str(best_combination) + "\n")
    f.write("max_depth, learning_rate, n_estimators, colsample_bytree,
        subsample" + "\n")

(max_depth, learning_rate, n_estimators, colsample_bytree, subsample)
   = best_combination
xgboost = xgb.XGBRegressor(max_depth=max_depth, learning_rate=
    learning_rate, n_estimators=n_estimators,
                               colsample_bytree=colsample_bytree,
                                  subsample=subsample, objective='
                                  reg:squarederror')
xgboost.fit(Xtr, ytr)
with open(path_ + "best_estimator_xgb.pkl", 'wb') as f:
    pickle.dump(xgboost, f)

pred_tr = xgboost.predict(Xtr)
t0=time()
for i in range(10):
    pred_te = xgboost.predict(Xte)
tte=(time()-t0)/10
# get the feature importance from best_xgb and plot it
feature_importance = xgboost.feature_importances_
feature_importance = 100.0 * (feature_importance / feature_importance.
   max())
sorted_idx = argsort(feature_importance)
pos = arange(sorted_idx.shape[0]) + .5
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.yticks(pos, features_names[sorted_idx])
plt.savefig(path_ + "features_importance.png")
plt.close()

iso = argsort(acc_vl_all)
plt.plot(acc_tr_all[iso], 'x', label='Train')
plt.plot(acc_vl_all[iso], 'x', label='Validation')
plt.plot(acc_te_all[iso], 'x', label='Test')
plt.legend()
plt.xlabel('Model')
plt.ylabel('R2_Score')
plt.savefig(path_ + "XGBoost_tune_cv_history.png")
plt.close()

iso = argsort(acc_vl_all)[int(0.5*len(acc_vl_all)):]
plt.plot(acc_tr_all[iso], 'x', label='Train')
plt.plot(acc_vl_all[iso], 'x', label='Validation')
plt.plot(acc_te_all[iso], 'x', label='Test')
plt.legend()
plt.xlabel('Model')
plt.ylabel('R2_Score')
plt.savefig(path_ + "XGBoost_tune_cv_history_50perc.png")
plt.close()
```

```
        do_sensitivity(Xtr, features_names, target_name, xgboost.predict,
            __method__, path_sens)

        plot_mae_per_bin(ytr, yte, pred_te, target_name, __method__, path_)

        error_analysis(ytr,pred_tr,target_name,__method__,"Train",path_err)
        error_analysis(yte,pred_te,target_name,__method__,"Test",path_err)

        plot_target_vs_predicted(ytr, pred_tr, target_name, __method__, "Train
            ",path_)
        plot_target_vs_predicted(yte, pred_te, target_name, __method__, "Test"
            ,path_)
        export_metrics(ytr, pred_tr, yte, pred_te, __method__, ttr, tte,
            LOGISTIC_REGR, path_)

        print("See results in folder:", path_)

        # export_notebook_to_html()
        gather_all_ML_metrics(ROOT_DIR)

    except Exception as ex1:
        print(ex1)



def predict_xgboost(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR):
    __method__ = "XGBoost"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_pred = ROOT_DIR+"Predict"+os.sep+__method__+os.sep
    try:
        with open(path_ + "best_estimator_xgb.pkl", 'rb') as f:
            best_estimator_xgb = pickle.load(f)
    except Exception as e:
        print("Error: ", e)
        return

    pred_out = best_estimator_xgb.predict(Xout)

    # save predictions to file
    with open(path_pred + "Predictions_"+__method__+".csv", "w") as file:
        for yi in pred_out:
            file.write(str(yi) + '\n')

    plot_target_vs_predicted(yout, pred_out, target_name, __method__, "Out",
        path_pred)
    export_metrics_out(yout, pred_out, path_pred + __method__ + "_Out",
        LOGISTIC_REGR)
    error_analysis(yout, pred_out, target_name, __method__, "Out", path_pred)


    print("See results in folder: ", path_pred)


def do_QuantileGradientBoostingRegressor(ROOT_DIR, Xtr, Xte, ytr, yte,
```

```
target_name):
# open the text file
__method__ = "XGBoost"
path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
with open(path_ + 'Best_Combination.txt') as file:
    values = file.read()
values = values.split("\n")[0]
values = tuple(map(float, values.strip('()').split(', ')))
print("Best_Combination:",values)
# extract the parameters from the values
max_depth__ = int(values[0])
learning_rate__ = float(values[1])
n_estimators__ = int(values[2])
colsample_bytree__ = float(values[3])
subsample__ = float(values[4])

common_params = dict(
    learning_rate=learning_rate__,
    n_estimators=n_estimators__,
    max_depth=max_depth__,
    subsample = subsample__,
    max_features = colsample_bytree__
)

alpha_low = 0.05
print('Training Q'+str(round(100*alpha_low,0))+'%')
gbr = GradientBoostingRegressor(loss="quantile", alpha=alpha_low, **
    common_params)
model_low = gbr.fit(Xtr, ytr)

alpha_med = 0.5
print('Training Q'+str(round(100*alpha_med,0))+'%')
gbr = GradientBoostingRegressor(loss="quantile", alpha=alpha_med, **
    common_params)
model_med = gbr.fit(Xtr, ytr)

alpha_high = 0.95
print('Training Q'+str(round(100*alpha_high,0))+'%')
gbr = GradientBoostingRegressor(loss="quantile", alpha=alpha_high, **
    common_params)
model_high = gbr.fit(Xtr, ytr)

plt.figure(figsize=(200, 20))
plt.plot(yte, label=target_name + ' Test Set', marker='x')
plt.plot(model_low.predict(Xte), label='Q'+str(round(100*alpha_low,0))+'%'
    , marker='v')
plt.plot(model_med.predict(Xte), label='Q'+str(round(100*alpha_med,0))+'%'
    , marker='x')
plt.plot(model_high.predict(Xte), label='Q'+str(round(100*alpha_high,0))+'
    %', marker='^')
plt.grid()
plt.xticks(fontsize=50)
plt.yticks(fontsize=50)
plt.legend(fontsize=50)
```

```
    plt.tight_layout()
    plt.savefig(path_ + "QuantileGradientBoostingRegressor_Test.png")
    plt.close()

    # Save the trained model to a file
    with open(path_ + "model_low.pkl", "wb") as f:
        pickle.dump(model_low, f)
    # Save the trained model to a file
    with open(path_ + "model_med.pkl", "wb") as f:
        pickle.dump(model_med, f)
    # Save the trained model to a file
    with open(path_ + "model_high.pkl", "wb") as f:
        pickle.dump(model_high, f)

def predict_quantile_gb(ROOT_DIR, Xout):
    __method__ = "XGBoost"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_pred = ROOT_DIR+"Predict"+os.sep+__method__+os.sep
    # Load the trained model_low from the file
    with open(path_ + "model_low.pkl", "rb") as f:
        model_low = pickle.load(f)
    # Use the loaded model to make predictions on new data
    y_pred_low = model_low.predict(Xout)

    # Load the trained model_med from the file
    with open(path_ + "model_med.pkl", "rb") as f:
        model_med = pickle.load(f)
    # Use the loaded model to make predictions on new data
    y_pred_med = model_med.predict(Xout)

    # Load the trained model_high from the file
    with open(path_ + "model_high.pkl", "rb") as f:
        model_high = pickle.load(f)
    # Use the loaded model to make predictions on new data
    y_pred_high = model_high.predict(Xout)

    # Create a DataFrame with the predictions
    df = pd.DataFrame({ 'y_pred_low': y_pred_low,
                        'y_pred_med': y_pred_med,
                        'y_pred_high': y_pred_high})
    # Save the DataFrame to an Excel file
    df.to_excel(path_pred + 'predict_quantile_gb.xlsx', index=False)
```

21 Appendix: Train Random Forests - ml_random_forests.py

```
# Description: This file contains the code for training and evaluating rf
    models.


from import_libraries import *
from misc_functions import *


def do_random_forests(Xtr,Xte,ytr,yte,features_names,target_name,
    __thres_early_stop__,__thres_min_tune_rounds__,PERMUTE_TRAIN_TEST,
    LOGISTIC_REGR,ROOT_DIR):
     __n_jobs__ = 4
     try:
         t0=time()
         # This parameter determines the number of decision trees to build.
             Increasing the number of trees typically
         # leads to better performance, but also increases the computational
             complexity and training time.
         n_estimators_ = list(concatenate((arange(1,11),arange(20, 101, 10),
             arange(200, 1001, 100))))

         # This parameter controls the maximum depth of each decision tree.
             Increasing the maximum depth can improve
         # the model's ability to fit complex patterns in the data, but can
             also lead to overfitting.
         max_depth_ = list(arange(1, 11))

         # This parameter controls the minimum number of samples required to
             split an internal node.
         # Increasing this parameter can prevent overfitting by requiring more
             samples to be present before a split is considered.
         min_samples_split_ = list(concatenate((arange(2,11),arange(20, 101,
             10))))

         # This parameter determines the number of features to consider when
             looking for the best split.
         # Increasing this parameter can improve the model's ability to capture
              complex patterns in the data, but can also increase overfitting.
         max_features_ = list(np_round(arange(0.1, 1.01, 0.05), 2))

         # This parameter controls the minimum number of samples required to be
              at a leaf node.
         # Increasing this parameter can prevent overfitting by requiring each
             leaf to have more samples.
         min_samples_leaf_ = list(concatenate((arange(1,11),arange(20, 101, 10)
             )))


         combinations = list(product(n_estimators_, max_depth_,
             min_samples_split_, max_features_, min_samples_leaf_))
         # randomly permute the combinations
```

```
combinations = shuffle(combinations, random_state=0)


# split train to train and validation
___perc_cv___ = 0.8; nof_folds = 5; obs = len(ytr)
tr_inds, vl_inds = split_tr_vl(obs,___perc_cv___,nof_folds,
    PERMUTE_TRAIN_TEST)

# train rf on each combination and evaluate on validation set
acc_tr_all = array([]); acc_vl_all = array([]); acc_te_all = array([])
opti_tr = -1; opti_vl = -1; opti_te= -1
best_combination = None
for i, (n_estimators_, max_depth_, min_samples_split_, max_features_,
    min_samples_leaf_) in enumerate(combinations):
    # catch exception if the combination is not valid
    try:
        print("Training Random Forests Model",i+1,"/>",
            __thres_min_tune_rounds__,"/",len(combinations),":",
                                    (n_estimators_, max_depth_,
                                        min_samples_split_,
                                        max_features_,
                                        min_samples_leaf_))
        rf = RandomForestRegressor(random_state=0, n_estimators =
            n_estimators_, max_depth = max_depth_,
                                    min_samples_split =
                                        min_samples_split_,
                                        max_features = max_features_,
                                    min_samples_leaf =
                                        min_samples_leaf_, n_jobs =
                                        __n_jobs__)
        acc_tr = 0; acc_vl = 0; acc_te = 0
        for fold in range(nof_folds):
            rf.fit(Xtr[tr_inds[fold],:], ytr[tr_inds[fold]])
            pred_tr = rf.predict(Xtr[tr_inds[fold],:])
            pred_vl = rf.predict(Xtr[vl_inds[fold],:])
            pred_te = rf.predict(Xte)
            acc_tr += pearsonr(ytr[tr_inds[fold]], pred_tr).
                correlation
            acc_vl += pearsonr(ytr[vl_inds[fold]], pred_vl).
                correlation
            acc_te += pearsonr(yte, pred_te).correlation
        acc_tr /= nof_folds; acc_vl /= nof_folds; acc_te /= nof_folds
        acc_tr_all = np_append(acc_tr_all, acc_tr)
        acc_vl_all = np_append(acc_vl_all, acc_vl)
        acc_te_all = np_append(acc_te_all, acc_te)
        # print all the r2 scores
        vl_75 = percentile(acc_vl_all, 75)
        vl_max = max(acc_vl_all)
        slope = (vl_max - vl_75)/(0.25*(i+1))
        print(datetime.datetime.now().strftime("%H:%M:%S"),"Comb:",i,"
            R2 Score:: Train:",acc_tr,"Validation:",acc_vl,"Test:",
            acc_te,
             "\nMax Val:",vl_max,"Q75 Val:",vl_75,"Slope:",slope)
        imax = argmax(acc_vl_all)
```

```
            best_combination = combinations[imax]
            print(best_combination)
            if i>__thres_min_tune_rounds__ and slope <__thres_early_stop__
                :
                print("Early Stopping, vl_max:",vl_max,"vl_95:",vl_75)
                break
        except:
            break

# find minimum length among acc_tr_all, acc_vl_all, acc_te_all, in
    case of exception
max_len = min([len(acc_tr_all), len(acc_vl_all), len(acc_te_all)])
# make all the arrays of same length
acc_tr_all = acc_tr_all[:max_len]
acc_vl_all = acc_vl_all[:max_len]
acc_te_all = acc_te_all[:max_len]

imax = argmax(acc_vl_all)
best_combination = combinations[imax]
print("Best Combination:",best_combination)

ttr = time()-t0
__method__ = "RF"
path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
path_err = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
    Error_Analysis"+os.sep
path_sens = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
    Sensitivity_Analysis"+os.sep

(n_estimators_, max_depth_, min_samples_split_, max_features_,
    min_samples_leaf_) = best_combination
rf = RandomForestRegressor(random_state=0, n_estimators =
    n_estimators_, max_depth = max_depth_,
                               min_samples_split = min_samples_split_,
                                   max_features = max_features_,
                               min_samples_leaf = min_samples_leaf_)
rf.fit(Xtr, ytr)
with open(path_ + "best_estimator_rf.pkl", 'wb') as f:
    pickle.dump(rf, f)

pred_tr = rf.predict(Xtr)
t0=time()
for i in range(10):
    pred_te = rf.predict(Xte)
tte=(time()-t0)/10
# get the feature importance from best_xgb and plot it
feature_importance = rf.feature_importances_
feature_importance = 100.0 * (feature_importance / feature_importance.
    max())
sorted_idx = argsort(feature_importance)
pos = arange(sorted_idx.shape[0]) + .5
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align='center')
```

```
        plt.yticks(pos, features_names[sorted_idx])
        plt.savefig(path_ + "features_importance.png")
        plt.close()

        iso = argsort(acc_vl_all)
        plt.plot(acc_tr_all[iso], 'x', label='Train')
        plt.plot(acc_vl_all[iso], 'x', label='Validation')
        plt.plot(acc_te_all[iso], 'x', label='Test')
        plt.legend()
        plt.xlabel('Model')
        plt.ylabel('R2_Score')
        plt.savefig(path_ + "rf_tune_cv_history.png")
        plt.close()

        iso = argsort(acc_vl_all)[int(0.5*len(acc_vl_all)):]
        plt.plot(acc_tr_all[iso], 'x', label='Train')
        plt.plot(acc_vl_all[iso], 'x', label='Validation')
        plt.plot(acc_te_all[iso], 'x', label='Test')
        plt.legend()
        plt.xlabel('Model')
        plt.ylabel('R2_Score')
        plt.savefig(path_ + "rf_tune_cv_history_50perc.png")
        plt.close()

        do_sensitivity(Xtr, features_names, target_name, rf.predict,
            __method__, path_sens)

        plot_mae_per_bin(ytr, yte, pred_te, target_name, __method__, path_)

        error_analysis(ytr,pred_tr,target_name,__method__,"Train",path_err)
        error_analysis(yte,pred_te,target_name,__method__,"Test",path_err)

        plot_target_vs_predicted(ytr, pred_tr, target_name, __method__, "Train
            ",path_)
        plot_target_vs_predicted(yte, pred_te, target_name, __method__, "Test"
            ,path_)
        export_metrics(ytr, pred_tr, yte, pred_te, __method__, ttr, tte,
            LOGISTIC_REGR, path_)

        print("See_results_in_folder:", path_)

        # export_notebook_to_html()
        gather_all_ML_metrics(ROOT_DIR)

    except Exception as ex1:
        print(ex1)



def predict_rf(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR):
    __method__ = "RF"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_pred = ROOT_DIR+"Predict"+os.sep+__method__+os.sep
```

```
try:
    with open(path_ + "best_estimator_rf.pkl", 'rb') as f:
        best_estimator_rf = pickle.load(f)
except Exception as e:
    print("Error: ", e)
    return

pred_out = best_estimator_rf.predict(Xout)

#  save predictions to file
with open(path_pred+"Predictions_"+__method__+".csv", "w") as file:
    for yi in pred_out:
        file.write(str(yi) + '\n')

plot_target_vs_predicted(yout, pred_out, target_name, __method__, "Out",
    path_pred)
export_metrics_out(yout, pred_out, path_pred + __method__ + "_Out",
    LOGISTIC_REGR)
error_analysis(yout, pred_out, target_name, __method__, "Out", path_pred)

print("See results in folder: ", path_pred)
```

22 Appendix: Train ANNBN - ml_ANNBN.py

```
# from IPython import get_ipython
from import_libraries import *
from misc_functions import *


def solve_bak(x, y, a):
    e = x@a - y
    err_all = array([])
    for j in range(10):
        for i in range(x.shape[1]):
            da = sum(e*x[:,i])/dot(x[:,i],x[:,i])
            e -= da*x[:,i]
            a[i] -= da
        err_all = np_append(err_all, mean(abs(e)))
    return a

# normalize ytr to have minimum -0.99 and maximum 0.99
def normalize_y(ytr):
    min_y = min(ytr)
    ytr = ytr - min_y
    max_y = max(ytr)
    ytr = ytr / max_y
    ytr = ytr * 0.98
    ytr = ytr + 0.01
    ytr = ytr * 2
    ytr = ytr - 1
    return ytr, min_y, max_y

#  denormalize ytr to their initial values
def denormalize_y(ytr, min_y, max_y):
    ytr = ytr + 1
    ytr = ytr / 2
    ytr -= 0.01
    ytr /= 0.98
    ytr *= max_y
    ytr += min_y
    return ytr

def compute_w(Xtr,ytr,neurons,min_obs_over_neurons):
    i_train = Xtr.shape[0]
    layer1 = zeros((i_train, neurons))
    w_all = []
    i_err = []
    ii_all = []
    n_train_internal = int(2*Xtr.shape[0]/min_obs_over_neurons)

    # k means clustering of Xtr
    # print("K-means clustering running for",neurons,"neurons.")
    #  normalize Xtr
    Xclust = (Xtr-mean(Xtr,axis=0))
    Xclust = Xclust/std(Xclust,axis=0)
    kmeans = KMeans(n_clusters=neurons,random_state=0,init='k-means++',n_init=
```

```
        'auto').fit(Xclust)
    # print("K-means clustering done.")

    # print("Computing internal neurons' weights...")
    iiXtr = arange(Xtr.shape[0])
    for i in range(neurons):
        # ii = permutation(Xtr.shape[0])[:n_train_internal]
        ii = iiXtr[kmeans.labels_==i]
        ii_all.append(ii)
        try:
            aa = lstsq(Xtr[ii],arctanh(ytr[ii]), rcond=None)[0]
            w_all.append(aa)
            layer1[:, i] = tanh(dot(Xtr, aa))
        except Exception as ex:
            i_err.append(i)
            print("Error in neuron ", i, ":", ex)
        # if i in range(0, neurons, int(math.floor(neurons/10))) or i ==
            neurons:
        #     print("Training ", i, " of ", neurons, " internal neurons.")


    i_keep = delete(arange(neurons), i_err)
    layer1 = layer1[:, i_keep]
    neurons = len(i_keep)
    ii_all = [ii_all[i] for i in i_keep]

    idx = arange(neurons)
    # print("SVD running...")
    # u, s, v = svd(layer1)
    # k = sum(s > 1e-10)
    # print("SVD done. k =", k,"of",layer1.shape[1],"neurons.")
    # print("QR-Factorization running...")
    # q, r, p = linalg.qr(layer1, pivoting=True)
    # idx = p[:k]
    # layer1 = layer1[:, idx]
    # neurons = len(idx)
    # w_all = [w_all[i] for i in idx]
    # ii_all = [ii_all[i] for i in idx]

    return layer1, w_all, ii_all, neurons, idx

def compute_layer1_te(Xte):
    global w_all, V
    # neurons = len(w_all)
    # layer1_te = zeros((Xte.shape[0], neurons))
    # for i in range(neurons):
    #     layer1_te[:,i] = tanh(dot(Xte, w_all[i]))
    layer1_te = tanh(dot(Xte, w_all.T))
    return layer1_te

def pred_annbn(Xpred):
    global w_all, V
    return dot(tanh(dot(Xpred, w_all.T)), V)
```

```
def do_ANNBN(Xtr, Xte, ytr, yte, features_names, target_name,
    PERMUTE_TRAIN_TEST, LOGISTIC_REGR, ROOT_DIR, min_obs_over_neurons):
    global w_all, V

    try:
        neurons=int(Xtr.shape[0]/min_obs_over_neurons)

        t0=time()
        # ytr, min_y, max_y = normalize_y(ytr)
        # layer1, w_all, ii_all, neurons, idx = compute_w(Xtr,ytr,neurons,
            min_obs_over_neurons)
        # w_all = array(w_all)
        # ytr = denormalize_y(ytr, min_y, max_y)
        # layer1_te = compute_layer1_te(Xte)

        ___perc_cv___ = 0.8; nof_folds = 5; obs = len(ytr)
        tr_inds, vl_inds = split_tr_vl(obs,___perc_cv___,nof_folds,
            PERMUTE_TRAIN_TEST)
        opti_idx = None
        acc_tr_all = array([]); acc_vl_all = array([]); acc_te_all = array([])
        i_all = array([]).astype(int)
        # step = int(Xtr.shape[0]/50)
        step = int(neurons/20)
        ki= 'rank'
        list_run = list(range(10, neurons, step))
        list_run.append(neurons)
        list_run = unique(list_run)[::-1]
        for i in list_run:
            try:
                ytr, min_y, max_y = normalize_y(ytr)
                layer1, w_all, ii_all, _, idx = compute_w(Xtr,ytr,i,
                    min_obs_over_neurons)
                w_all = array(w_all)
                ytr = denormalize_y(ytr, min_y, max_y)
                layer1_te = compute_layer1_te(Xte)

                inds_keep = range(i)#idx[:i]
                acc_tr = 0; acc_vl = 0; acc_te = 0
                V = zeros(i)
                print("Solving_for",i,"neurons.")
                for fold in range(nof_folds):
                    # res = regression_bak(layer1[tr_inds[fold],:][:,inds_keep
                        ], ytr[tr_inds[fold]])
                    # if res.flag != "OK":
                    #     raise Exception("Regression failed at",i,"neurons.")
                    # V = res.aa
                    V,_,ki,_ = lstsq(layer1[tr_inds[fold],:][:,inds_keep], ytr
                        [tr_inds[fold]], rcond=None)
                    # X = layer1[tr_inds[fold],:][:,inds_keep]
                    # XX = X.T@X
                    # V = solve(XX, X.T@ytr[tr_inds[fold]])
                    # V = solve_bak(layer1[tr_inds[fold],:][:,inds_keep], ytr[
                        tr_inds[fold]], V)
```

```
            pred_tr = dot(layer1[tr_inds[fold],:][:,inds_keep], V)
            pred_vl = dot(layer1[vl_inds[fold],:][:,inds_keep], V)
            pred_te = dot(layer1_te[:,inds_keep], V)
            acc_tr += pearsonr(ytr[tr_inds[fold]], pred_tr).
                correlation
            acc_vl += pearsonr(ytr[vl_inds[fold]], pred_vl).
                correlation
            acc_te += pearsonr(yte, pred_te).correlation
        acc_tr /= nof_folds; acc_vl /= nof_folds; acc_te /= nof_folds
        acc_tr_all = np_append(acc_tr_all, acc_tr)
        acc_vl_all = np_append(acc_vl_all, acc_vl)
        acc_te_all = np_append(acc_te_all, acc_te)
        i_all = np_append(i_all, i)
        print(datetime.datetime.now().strftime("%H:%M:%S"),"
            CombNeurons:",i,"of",neurons,". Rank:",ki,
            "\nR2 Score:: Train:",acc_tr,"Validation:",acc_vl,"Test:",
                acc_te)
    except KeyboardInterrupt:
        print('KeyboardInterrupt: Stopped by user')
        break

# find minimum length among acc_tr_all, acc_vl_all, acc_te_all, in
    case of exception
max_len = min([len(acc_tr_all), len(acc_vl_all), len(acc_te_all)])
# make all the arrays of same length
acc_tr_all = acc_tr_all[:max_len]
acc_vl_all = acc_vl_all[:max_len]
acc_te_all = acc_te_all[:max_len]

imax = argmax(acc_vl_all)
imax = i_all[imax]
print(">>>>>>>>>>>>>>>>>>>>Optimal number of neurons:",imax,"
    <<<<<<<<<<<<<<<<<<<<")
# opti_idx = range(imax)
# layer1 = layer1[:, opti_idx]
# layer1_te = layer1_te[:, opti_idx]
# neurons = len(opti_idx)
# w_all = w_all[opti_idx,:]
# ii_all = [ii_all[i] for i in opti_idx]

neurons = imax.copy()
ytr, min_y, max_y = normalize_y(ytr)
layer1, w_all, ii_all, neurons, idx = compute_w(Xtr,ytr,imax,
    min_obs_over_neurons)
w_all = array(w_all)
ytr = denormalize_y(ytr, min_y, max_y)
layer1_te = compute_layer1_te(Xte)


ttr = time()-t0
__method__ = "ANNBN"
path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
path_err = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
    Error_Analysis"+os.sep
```

```
path_sens = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
    Sensitivity_Analysis"+os.sep

cdf_pdf_plot(layer1.flatten(),"Layer1Histogram",path_)

# res = regression_bak(layer1, ytr)
# V = res.aa
V,_,ki,_ = lstsq(layer1, ytr, rcond=None)
pred_tr = dot(layer1, V)
t0=time()
for i in range(10):
    # layer1_te = compute_layer1_te(Xte)#####
    # pred_te = dot(layer1_te, V)
    pred_annbn(Xte)
tte=(time()-t0)/10

try:
    export_equation_in_excel(Xtr, ytr, features_names, target_name,
        min_y, max_y, w_all, V, path_, ROOT_DIR)
except Exception as ex:
    print("Error in exporting equation:", ex)
    return

with open(path_ + "NeuronsWeights.csv", "w") as file:
    for wi in w_all:
        for wij in wi:
            file.write(str(wij) + ',')
        file.write('\n')
with open(path_ + "ExternalLayerWeights.csv", "w") as file:
    for Vi in V:
        file.write(str(Vi) + ',')

iso = argsort(acc_vl_all)
plt.plot(i_all[iso], acc_tr_all[iso], 'x', label='Train')
plt.plot(i_all[iso], acc_vl_all[iso], 'x', label='Validation')
plt.plot(i_all[iso], acc_te_all[iso], 'x', label='Test')
plt.legend()
plt.xlabel('Neurons')
plt.ylabel('R2_Score')
plt.savefig(path_ + "ANNBN_tune_cv_history.png")
plt.close()

iso = argsort(acc_vl_all)[:argmax(acc_vl_all)]
plt.plot(i_all[iso], acc_tr_all[iso], 'x', label='Train')
plt.plot(i_all[iso], acc_vl_all[iso], 'x', label='Validation')
plt.plot(i_all[iso], acc_te_all[iso], 'x', label='Test')
plt.legend()
plt.xlabel('Neurons')
plt.ylabel('R2_Score')
plt.savefig(path_ + "ANNBN_tune_cv_history_imax.png")
plt.close()

do_sensitivity(Xtr, features_names, target_name, pred_annbn,
    __method__, path_sens)
```

```
        plot_mae_per_bin(ytr, yte, pred_te, target_name, __method__, path_)

        error_analysis(ytr,pred_tr,target_name,__method__,"Train",path_err)
        error_analysis(yte,pred_te,target_name,__method__,"Test",path_err)

        plot_target_vs_predicted(ytr, pred_tr, target_name, __method__, "Train
            ",path_)
        plot_target_vs_predicted(yte, pred_te, target_name, __method__, "Test"
            ,path_)
        export_metrics(ytr, pred_tr, yte, pred_te, __method__, ttr, tte,
            LOGISTIC_REGR, path_)

        print("See results in folder:", path_)

        # export_notebook_to_html()
        gather_all_ML_metrics(ROOT_DIR)

    except Exception as ex1:
        print(ex1)
```

```
def export_equation_in_excel(Xtr, ytr, features_names, target_name, min_y,
    max_y, w_all, V, path_, ROOT_DIR):
    col_excel = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "
        M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z"]
    col_excel0 = list(col_excel)
    col_excel = col_excel + [f'A{c}' for c in col_excel0]
    col_excel = col_excel + [f'B{c}' for c in col_excel0]
    col_excel = col_excel + [f'C{c}' for c in col_excel0]
    col_excel = col_excel + [f'D{c}' for c in col_excel0]
    col_excel = array(col_excel)

    inds_exclude = []
    if os.path.isfile(ROOT_DIR + "excluded_features.txt"):
        with open(ROOT_DIR + "excluded_features.txt", 'r') as f:
            inds_exclude = [int(line.strip()) for line in f]
        inds_exclude = array(inds_exclude)
        print("Multicolinearity was detected, excluding columns from excel: ",
            [i for i in col_excel[inds_exclude]])
    col_excel = delete(col_excel, inds_exclude, axis=0)

    equation_excel = "="
    minYstr = str(min_y)
    if min_y > 0:
        minYstr = "+" + minYstr
    nof_files = 1
    equations_all = []
    neurons = len(w_all)
    for i in range(neurons):
```

```
    new_feature = ""
    for j in range(Xtr.shape[1]):
        str_ = "{:.15E}".format(w_all[i][j])
        if w_all[i][j] >= 0:
            str_ = "+" + str_
        new_feature = "{}{}*{}2".format(new_feature, str_, col_excel[j])

    str00 = "{:.15E}".format(V[i])
    if V[i] >= 0:
        str00 = "+" + str00
    new_feature = "{}*tanh({})".format(str00, new_feature)

    if len(equation_excel) + len(new_feature) > 8000:
        equations_all.append(equation_excel)
        nof_files += 1
        equation_excel = "="
        print("Equation split at neuron:", i)

    equation_excel = "{}{}".format(equation_excel, new_feature)

equations_all.append(equation_excel)
print(">>>>>>>>>>>>>>>>>>>>equation_excel Done in", nof_files, "parts
    .<<<<<<<<<<<<<<<")

i_med = argsort(ytr)[int(round(len(ytr) / 2))]
xx_tr_med_given_init = c_[Xtr[i_med, newaxis], array([ytr[i_med]])]
# Number of columns in the original matrix
n_cols = xx_tr_med_given_init.shape[1]
# Number of new columns to insert
n_new_cols = len(inds_exclude)
# Initialize a matrix of zeros with the new shape
new_matrix = zeros((1, n_cols + n_new_cols))
# Copy the original matrix into the new matrix, excluding the columns to
    be excluded
new_matrix[:, [i for i in range(n_cols + n_new_cols) if i not in
    inds_exclude]] = xx_tr_med_given_init
xx_tr_med_given_init = new_matrix

equations_all_mat = empty((1, len(equations_all)), dtype=object)
pred_str = "="
for i in range(len(equations_all)):
    equations_all_mat[0, i] = equations_all[i]
    pred_str = pred_str + col_excel[xx_tr_med_given_init.shape[1] - len(
        inds_exclude) + i] + "2+"
pred_str = pred_str[:-1]
equations_all_mat = column_stack([equations_all_mat, pred_str])
equations_all_mat = column_stack([xx_tr_med_given_init, equations_all_mat
    ])
df = pd.DataFrame(equations_all_mat)

# New label to insert
new_label = 'collinear'
# Initialize a new Index object with the same labels as features_names
new_index = features_names.copy()
```

```
    # Insert the new label at the specified positions
    for i in sorted(inds_exclude):
        new_index = new_index.insert(i, new_label)
    excel_col_pred_names = np_append(new_index, [target_name, *[f"pred-{i}"
        for i in range(1, len(equations_all) + 1)], "Prediction"])
    df.rename(columns=dict(zip(df.columns, excel_col_pred_names)), inplace=
        True)
    df.to_excel(path_ + "equation_excel.xlsx", index=False, engine="openpyxl")


def predict_ANNBN(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR):
    global w_all, V, cwd
    __method__ = "ANNBN"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_pred = ROOT_DIR+"Predict"+os.sep+__method__+os.sep
    # try to load the results from the CSV file
    try:
        with open(path_ + "NeuronsWeights.csv", 'r') as f:
            reader = csv.reader(f)
            w_all = list(reader)
        # delete the last element, of each row, which is an empty string
        for wi in w_all:
            wi.pop()
        w_all = [[float(wij) for wij in wi] for wi in w_all]
        w_all = array(w_all)

        with open(path_ + "ExternalLayerWeights.csv", 'r') as f:
            reader = csv.reader(f)
            V = list(reader)
        V[0].pop()
        V = [float(Vi) for Vi in V[0]]

        neurons = len(w_all)
        print("Loaded ANNBN weights from file")

        pred_out = pred_annbn(Xout)
    except Exception as e:
        print("Error: ", e)
        return

    # save predictions to file
    with open(path_pred+"Predictions_"+__method__+".csv", "w") as file:
        for yi in pred_out:
            file.write(str(yi) + '\n')

    plot_target_vs_predicted(yout, pred_out, target_name, __method__, "Out",
        path_pred)
    export_metrics_out(yout, pred_out, path_pred + __method__ + "_Out",
        LOGISTIC_REGR)
    error_analysis(yout, pred_out, target_name, __method__, "Out", path_pred)

    print("See results in folder: ", path_pred)
```

23 Appendix: Train DANN - ml_DANN.py

```
from import_libraries import *
from misc_functions import *

class torch_set_dataset(Dataset):
    def __init__(self, X_data, y_data):
        self.X_data = X_data
        self.y_data = y_data
    def __getitem__(self, index):
        return self.X_data[index], self.y_data[index]
    def __len__ (self):
        return len(self.X_data)


class MultipleRegression(nn.Module):
    def __init__(self, num_features, num_targets, neurons, dropout, layers):
        super(MultipleRegression, self).__init__()
        self.num_features = num_features
        self.num_targets = num_targets
        self.layers = layers
        self.dropout = dropout
        self.linear_layers = nn.ModuleList()
        # Add input layer
        self.linear_layers.append(nn.Linear(num_features, neurons))
        torch.nn.init.xavier_uniform_(self.linear_layers[-1].weight)
        self.linear_layers.append(nn.ReLU())
        self.linear_layers.append(nn.Dropout(p=dropout))
        # Add hidden layers
        for i in range(layers-1):
            self.linear_layers.append(nn.Linear(neurons, neurons))
            torch.nn.init.xavier_uniform_(self.linear_layers[-1].weight)
            self.linear_layers.append(nn.ReLU())
            self.linear_layers.append(nn.Dropout(p=dropout))
        # Add output layer
        self.linear_layers.append(nn.Linear(neurons, num_targets))
        torch.nn.init.xavier_uniform_(self.linear_layers[-1].weight)
    def forward(self, x):
        for layer in self.linear_layers:
            x = layer(x)
        return x


def plot_all_losses(all_losses_tr,all_losses_vl):
    # convert all_losses_vl to numpy array
    all_losses_vl = array(all_losses_vl)
    all_losses_tr = array(all_losses_tr)
    iso = argsort(all_losses_vl)[::-1]
    plt.plot(all_losses_tr[iso], label='Train Losses')
    plt.plot(all_losses_vl[iso], label='Validation Losses')
    plt.legend(loc='upper right')
    plt.tight_layout()
```

```
    plt.savefig("all_losses_tuning.png")
    plt.close()


def plot_history(history,perc_of_epochs_to_plot,tag1,tag2,path_):
    nn = int(len(history['train'])*(100-perc_of_epochs_to_plot)/100)
    # plot train and validation loss
    plt.plot((array(history['train'][nn:]))**0.5, label=tag1)
    plt.plot((array(history['val'][nn:]))**0.5, label=tag2)
    plt.title('RMSE')
    plt.legend(loc='upper_right')
    plt.tight_layout()
    plt.savefig(path_ + "loss_history_percentage_kept_"+str(
        perc_of_epochs_to_plot)+".png")
    plt.close()

def pred_DANN(XX):
    global model

    # Set the model to evaluation mode
    model.eval()
    # Convert the input data to a PyTorch tensor
    XXX = torch.tensor(XX).float()
    # XXX = XXX.to('cuda')
    # Use the model to make predictions on the input data
    with torch.no_grad():
        pred_tr = model(XXX)
    XXX=0
    model.train()
    # Convert the predicted values to a NumPy array
    pred_tr = pred_tr.cpu().numpy()
    return pred_tr.reshape(-1)

def train_pytorch(XTR, YTR, XVL, YVL, params, path_, save_model=False):
    global model

    # params = combinations[0]
    # fold = 0
    # XTR = Xtr[tr_inds[fold],:]; YTR = ytr[tr_inds[fold]]
    # XVL = Xtr[vl_inds[fold],:]; YVL = ytr[vl_inds[fold]]

    YTR = YTR.reshape(-1,1)
    YVL = YVL.reshape(-1,1)
    layers = params[0]; neurons = params[1]; epochs = params[2]
    learning_rate = params[3]; dropout = params[4]; batch = params[5];
        moment_um = params[6]

    train_dataset = torch_set_dataset(torch.from_numpy(XTR).float(), torch.
        from_numpy(YTR).float())
    val_dataset = torch_set_dataset(torch.from_numpy(XVL).float(), torch.
        from_numpy(YVL).float())

    train_loader = DataLoader(dataset=train_dataset, batch_size=batch, shuffle
        =False)
```

```
val_loader = DataLoader(dataset=val_dataset, batch_size=batch, shuffle=
    False)

model = MultipleRegression(XTR.shape[1], YTR.shape[1], neurons, dropout,
    layers)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
# print(device)
if torch.cuda.device_count() > 1:
    # print("Let's use", torch.cuda.device_count(), "GPUs!")
    model = nn.DataParallel(model)
model.to(device)
# if torch.cuda.is_available():
#     for i in range(torch.cuda.device_count()):
        # print(torch.cuda.get_device_name(i))

criterion = nn.MSELoss()

optimizer = optim.NAdam(model.parameters(), lr=learning_rate,
    momentum_decay=moment_um)

loss_stats = {'train': [], "val": []}

# print("Begin training.")
# t1=time()
for e in range(1, epochs+1):
    # TRAINING
    model.train()
    for X_train_batch, y_train_batch in train_loader:
        X_train_batch, y_train_batch = X_train_batch.to(device),
            y_train_batch.to(device)
        optimizer.zero_grad()
        y_train_pred = model(X_train_batch)
        train_loss = criterion(y_train_pred, y_train_batch)
        train_loss.backward()
        optimizer.step()

    loss_stats['train'].append(sqrt(mean((pred_DANN(XTR)-YTR.reshape(-1))
        **2)))
    loss_stats['val'].append(sqrt(mean((pred_DANN(XVL)-YVL.reshape(-1))
        **2)))
    # if e in range (1,epochs+1,1):
    #     print("==========",e,loss_stats['train'][-1],loss_stats['val
        '][-1])

# t2=time()
# t2_t1 = "%.1f" % (t2-t1)
# print("training got ",t2_t1," seconds")

if save_model:
    print("saving model...")
    torch.save(model, path_ + "model.pth")

return loss_stats, model
```

```
def do_DANN(Xtr, ytr, Xte, yte, features_names, target_name,
    __thres_early_stop__, __thres_min_tune_rounds__, PERMUTE_TRAIN_TEST,
    LOGISTIC_REGR, ROOT_DIR):
    global model
    __method__ = "DANN"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_err = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"Error_Analysis"+
        os.sep
    path_sens = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep+"
        Sensitivity_Analysis"+os.sep
    try:
        t0=time()

        scaler = MinMaxScaler()
        scaler.fit(Xtr)
        Xtr = scaler.transform(Xtr)
        Xte = scaler.transform(Xte)
        obs = Xtr.shape[0]

        torch.manual_seed(0)
        seed(0)
        random.seed(0)

        layers = [2, 5, 10]
        neurons = [10, 50, 100]
        epochs = list(concatenate((arange(1,11),arange(20, 101, 10),arange
            (200, 1001, 100))))
        learning_rate = [0.01, 0.001, 0.0001]
        dropout = [0.01, 0.05]
        batch = [int(obs/8), int(obs/4), int(obs/2)]
        moment_um = [0.25, 0.5, 0.75]
        combinations = list(product(layers, neurons, epochs, learning_rate,
            dropout, batch, moment_um))
        PARS = "layers, neurons, epochs, learning_rate, dropout, batch,
            moment_um"
        # randomly permute the combinations
        combinations = shuffle(combinations, random_state=0)

        # split train to train and validation
        ___perc_cv___ = 0.8; nof_folds = 5; obs = len(ytr)
        tr_inds, vl_inds = split_tr_vl(obs,___perc_cv___,nof_folds,
            PERMUTE_TRAIN_TEST)

        acc_tr_all = array([]); acc_vl_all = array([]); acc_te_all = array([])
        opti_tr = -1; opti_vl = -1; opti_te= -1
        best_combination = None
        for i,combination in enumerate(combinations):
            # catch exception if the combination is not valid
            try:
                print(datetime.datetime.now().strftime("%H:%M:%S"),"
                    ====================",
                    "Training DANN Model",i+1,"/>",__thres_min_tune_rounds__,"
                        /",len(combinations),":",combination)
```

```
        acc_tr = 0; acc_vl = 0; acc_te = 0
        for fold in range(nof_folds):
            history, model = train_pytorch(Xtr[tr_inds[fold],:], ytr[
                tr_inds[fold]],
                                           Xtr[vl_inds[fold],:], ytr[
                                               vl_inds[fold]],
                                               combination, path_, False)
            pred_tr = pred_DANN(Xtr[tr_inds[fold],:])
            pred_vl = pred_DANN(Xtr[vl_inds[fold],:])
            pred_te = pred_DANN(Xte)
            acc_tr += pearsonr(ytr[tr_inds[fold]], pred_tr).
                correlation
            acc_vl += pearsonr(ytr[vl_inds[fold]], pred_vl).
                correlation
            acc_te += pearsonr(yte, pred_te).correlation
        acc_tr /= nof_folds; acc_vl /= nof_folds; acc_te /= nof_folds
        acc_tr_all = np_append(acc_tr_all, acc_tr)
        acc_vl_all = np_append(acc_vl_all, acc_vl)
        acc_te_all = np_append(acc_te_all, acc_te)

        # print all the r2 scores
        vl_75 = percentile(acc_vl_all, 75)
        vl_max = max(acc_vl_all)
        slope = (vl_max - vl_75)/(0.25*(i+1))

        print(PARS, combination,
                "\nR2 Score:: Train:",acc_tr,"Validation:",acc_vl,"
                    Test:",acc_te,
                "\nMax Val:",vl_max,"Q75 Val:",vl_75,"Slope:",slope)

        imax = argmax(acc_vl_all)
        best_combination = combinations[imax]
        print(datetime.datetime.now().strftime("%H:%M:%S"),"
            ==========",
            "Best Combination:",best_combination)
        if i>__thres_min_tune_rounds__ and slope <__thres_early_stop__
            :
            print("Early Stopping, vl_max:",vl_max,"vl_95:",vl_75)
            break
    except KeyboardInterrupt:
        print('KeyboardInterrupt: Stopped by user')
        break

print("Best Combination:",best_combination)
# find minimum length among acc_tr_all, acc_vl_all, acc_te_all, in
    case of exception
max_len = min([len(acc_tr_all), len(acc_vl_all), len(acc_te_all)])
# make all the arrays of same length
acc_tr_all = acc_tr_all[:max_len]
acc_vl_all = acc_vl_all[:max_len]
acc_te_all = acc_te_all[:max_len]

ttr = time()-t0
# Save the scaler to a file
```

```
    with open(path_ + "scaler.pkl", "wb") as f:
        pickle.dump(scaler, f)

    history, model = train_pytorch(Xtr, ytr, Xte, yte, best_combination,
        path_, True)
    plot_history(history,100,'Train','Test',path_)
    plot_history(history,50,'Train','Test',path_)
    pred_tr = pred_DANN(Xtr)
    t0=time()
    for i in range(10):
        pred_te = pred_DANN(Xte)
    tte=(time()-t0)/10
    iso = argsort(acc_vl_all)
    plt.plot(acc_tr_all[iso], 'x', label='Train')
    plt.plot(acc_vl_all[iso], 'x', label='Validation')
    plt.plot(acc_te_all[iso], 'x', label='Test')
    plt.legend()
    plt.xlabel('Model')
    plt.ylabel('R2_Score')
    plt.savefig(path_ + "DANN_tune_cv_history.png")
    plt.close()

    iso = argsort(acc_vl_all)[int(0.5*len(acc_vl_all)):]
    plt.plot(acc_tr_all[iso], 'x', label='Train')
    plt.plot(acc_vl_all[iso], 'x', label='Validation')
    plt.plot(acc_te_all[iso], 'x', label='Test')
    plt.legend()
    plt.xlabel('Model')
    plt.ylabel('R2_Score')
    plt.savefig(path_ + "DANN_tune_cv_history_50perc.png")
    plt.close()

    do_sensitivity(Xtr, features_names, target_name, pred_DANN, __method__
        , path_sens)

    plot_mae_per_bin(ytr, yte, pred_te, target_name, __method__, path_)

    error_analysis(ytr,pred_tr,target_name,__method__,"Train",path_err)
    error_analysis(yte,pred_te,target_name,__method__,"Test",path_err)

    plot_target_vs_predicted(ytr, pred_tr, target_name, __method__, "Train
        ",path_)
    plot_target_vs_predicted(yte, pred_te, target_name, __method__, "Test"
        ,path_)
    export_metrics(ytr, pred_tr, yte, pred_te, __method__, ttr, tte,
        LOGISTIC_REGR, path_)

    print("See_results_in_folder:", path_)

    # export_notebook_to_html()
    gather_all_ML_metrics(ROOT_DIR)
except Exception as ex1:
    print(ex1)
```

```
def predict_DANN(Xout, yout, target_name, LOGISTIC_REGR, ROOT_DIR):
    global cwd, model
    __method__ = "DANN"
    path_ = ROOT_DIR+"ML_Models"+os.sep+__method__+os.sep
    path_pred = ROOT_DIR+"Predict"+os.sep+__method__+os.sep
    # try to load the results from the CSV file
    try:
        # Load the scaler from the file
        with open(path_ + "scaler.pkl", "rb") as f:
            scaler = pickle.load(f)
        Xout = scaler.transform(Xout)

        # load model
        model = torch.load(path_ + "model.pth")

        pred_out = pred_DANN(Xout)

    except Exception as e:
        print("Error: ", e)
        return

    # save predictions to file
    with open(path_pred + "Predictions_"+__method__+".csv", "w") as file:
        for yi in pred_out:
            file.write(str(yi) + '\n')

    plot_target_vs_predicted(yout, pred_out, target_name, __method__, "Out",
        path_pred)
    export_metrics_out(yout, pred_out, path_pred + __method__ + "_Out",
        LOGISTIC_REGR)
    error_analysis(yout, pred_out, target_name, __method__, "Out", path_pred)

    print("See results in folder: ", path_pred)
```