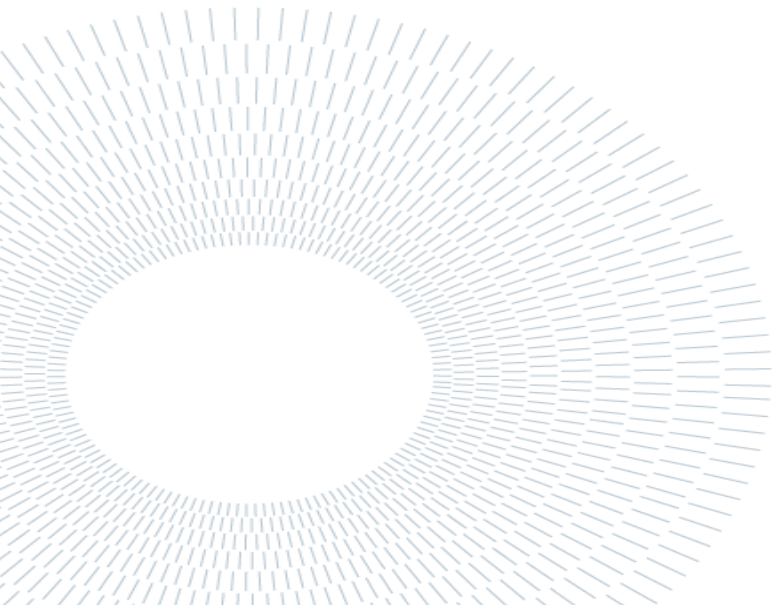
POLITECNICO
MILANO 1863

Winter Wheat Yield Mapping through the Integration of Earth Observation Data and Crop Model Simulations

MASTER THESIS IN "INGEGNERIA PER L'AMBIENTE E IL TERRITORIO, PIANIFICAZIONE E GESTIONE DELLE RISORSE NATURALI"

Author: **Alessandro Klingman**

Student ID: 10681596
Advisor: Chiara Corbari
Co-advisor: Héctor Nieto Solana
Academic Year: 2025-26



Abstract

This study proposes an approach for winter wheat yield mapping based on the integration of crop simulations and machine learning techniques. A large synthetic dataset was generated using the physically based Daisy crop model by varying soil parameters and agronomic management practices across multiple meteorological years, in order to represent a wide range of production conditions.

Correlation analysis and hierarchical clustering were applied to perform structured feature selection, reducing the dimensionality of the problem and identifying groups of variables characterized by similar relationships. Random Forest and Symbolic Regression models were then implemented to explore both non-linear ensemble approaches and interpretable symbolic formulations. The relative contribution of predictors within the models was also analyzed to assess their role in explaining yield variability.

In parallel, a set of variables realistically retrievable from Earth Observation data was constructed to evaluate the transferability of the approach to an operational context.

Results show high predictive performance within the synthetic domain and reduced accuracy when compared with observed data, suggesting the presence of structural differences between the simulated environment and real-world conditions. Overall, the study highlights the potential of integrating crop modelling and machine learning for yield mapping applications, while also emphasizing the challenges associated with transferring calibrated models to operational scenarios.

Key-words: Crop yield mapping; Crop simulation modelling; Machine learning; Earth Observations; Daisy; Copernicus

Abstract in italiano

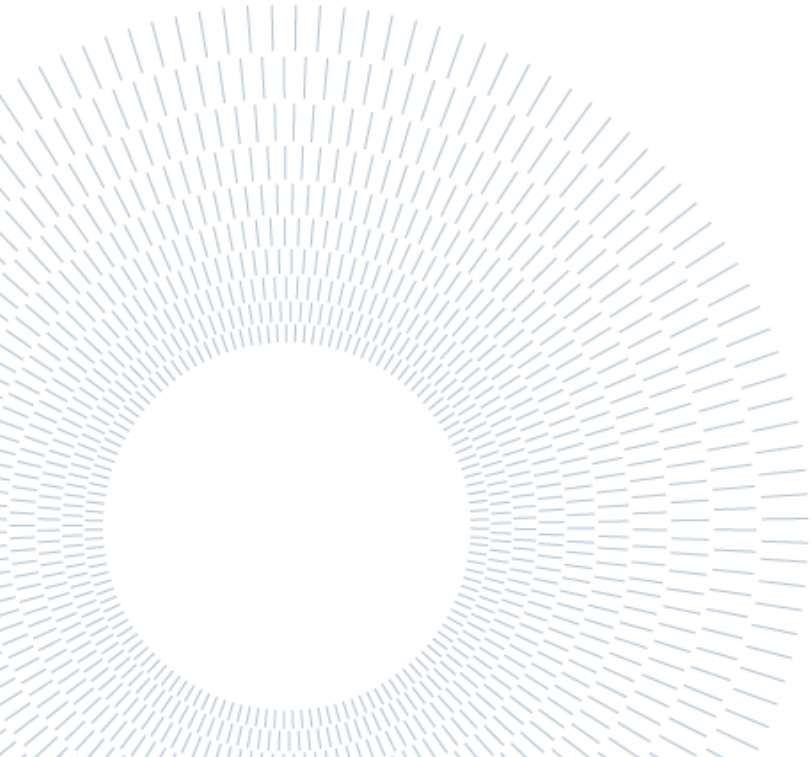
Questo lavoro propone un approccio per la mappatura della resa del grano invernale basato sull'integrazione tra simulazioni colturali e tecniche di machine learning. Un ampio dataset sintetico è stato generato mediante il modello fisicamente basato Daisy, variando parametri del suolo e della gestione agronomica su più anni meteorologici, al fine di rappresentare un ampio range di condizioni diverse.

Attraverso analisi di correlazione e clustering gerarchico è stata effettuata una selezione strutturata dei predittori, riducendo la dimensionalità del problema e individuando gruppi di variabili caratterizzate da relazioni simili. Su tali insiemi sono stati implementati modelli di Random Forest e Symbolic Regression, con l'obiettivo di esplorare sia approcci non lineari ensemble sia formulazioni simboliche interpretabili. È stato inoltre analizzato il peso relativo delle variabili all'interno dei modelli, al fine di evidenziarne il contributo nella spiegazione della resa.

Parallelamente, è stato costruito un insieme di variabili realisticamente derivabili da dati di Osservazione della Terra, per valutare la trasferibilità dell'approccio in un contesto operativo.

I risultati evidenziano elevate prestazioni nell'ambito del dataset sintetico e una riduzione dell'accuratezza nel confronto con dati osservati, suggerendo la presenza di differenze strutturali tra ambiente modellato e condizioni reali. Nel complesso, lo studio mette in luce le potenzialità dell'integrazione tra modellistica colturale e machine learning per applicazioni di mappatura della resa, evidenziandone al contempo le criticità nel passaggio verso scenari operativi.

Parole chiave: Mappatura della resa colturale; Modellistica di simulazione colturale; Machine learning; Osservazioni della Terra; Daisy; Copernicus



Contents

1 Introduction and related work

Crop yield estimates through remote sensing have been studied since the 1970s. Still, its first applications were not focused on estimating production on a single field scale, but rather on a regional level (MacDonald & Hall, 1980). More recent works have developed models on a field scale, capable of explaining more than half of yield variation (Lobell et al., 2005), but this kind of application has been limited for three reasons (Lobell et al., 2015):

1. The objective of funding entities is still to assess production at a larger level
2. Imagery acquisition and processing costs are often too high
3. Reliable accuracy has not been demonstrated across many contexts, because in situ data at field scale is more difficult to obtain than at aggregated administrative level, where governments regularly release survey-based statistics

Starting in the 2010s, improvements in data availability and processing platforms have substantially reduced the cost of acquiring and pre-processing satellite imagery, leading to a rapid increase in studies based on the large-scale processing of satellite data (M. C. Hansen et al., 2013).

Despite the recent progress in data availability, converting satellite observations into reliable yield estimates remains challenging. Traditional approaches often rely on empirical relationships between vegetation indices and yield, which generally require local calibration and may not generalize well across regions or years (Shanahan et al., 2001). Alternatively, approaches based on ecophysiological crop models can better account for variations in weather, management, and crop development, but they often involve computationally intensive data assimilation procedures and require detailed local inputs such as soil and daily meteorological data (Gallego et al., 2010).

An intermediate solution consists in using crop models to generate synthetic datasets that are subsequently employed to train simpler empirical or machine learning models, which can then be efficiently applied over large spatial and temporal scales (Clevers, 1997). However, such models often remain specific to the conditions under which they were calibrated.

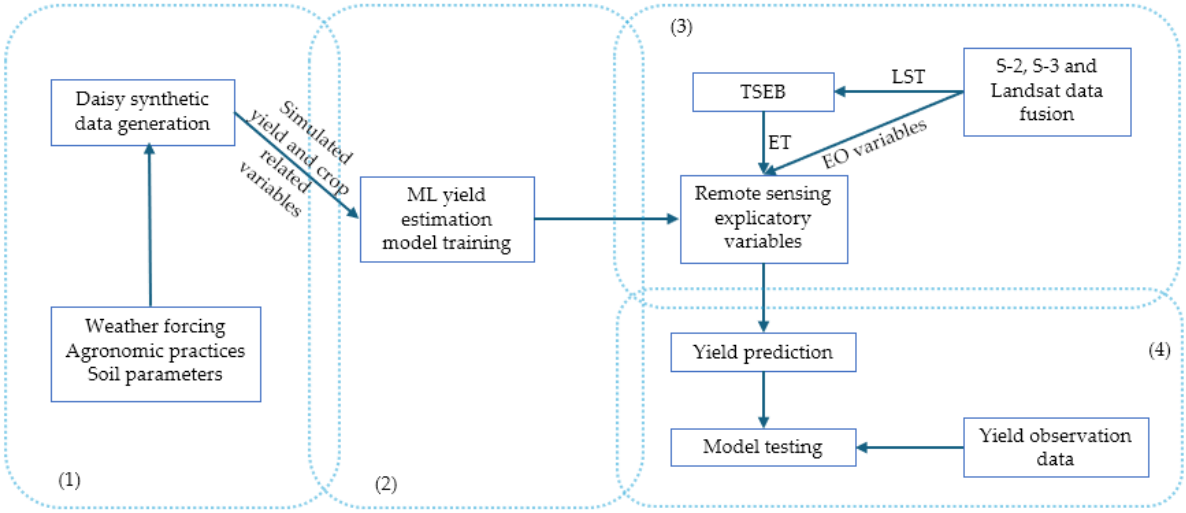


Figure 1: Workflow adopted in this study, consisting of (1) Daisy synthetic data generation, (2) ML model calibration, (3) remote sensing data retrieval and processing through S-2, S-3 and Landsat data fusion, (4) model validation using observed yield data

Lobell et al. [2015] developed a new way of approaching this technique of estimating yield, through a scalable satellite-based crop yield mapper (SCYM) (Lobell et al., 2015). The innovation of this method consists of including in the empirical model multiple image dates, which improves the adaptability to different sites, years and dates. In the work, APSIM (Agricultural Production Systems Simulator) model (Holzworth et al., 2014) was utilized to generate synthetically a total of 4200 simulations, spanning six sites, 14 years and a range of different management characteristics. Simulations were run both for maize and soybean fields in Midwestern United States, in the states of Iowa, Illinois and Indiana. Using synthetic data, a generic multiple linear regression was used to calibrate yield estimations parameters ($\beta_{i,d}$), based on weather-based predictors (W), such as monthly averages of rainfall, and a predictor variable of a remotely measured quantity on a specific date (RM_d).

$$Yield = \beta_{0,d} + \beta_{1,d} \cdot W + \beta_{2,d} \cdot RM_d + \beta_{3,d} \cdot W \cdot RM_d$$

Equation 1: Yield regression model (Lobell et al., 2015)

The remote sensing variable selected was a vegetation index, specifically the green chlorophyll vegetation index ($GCVI$), empirically computed with leaf area index (LAI) values. The tuned model was then validated with yield data reports provided by farmers for insurance purposes. Overall results of the SCYM approach show that it captures on average about one-third of field-scale yield variation for the area (Lobell et al., 2015).

The present study adopts a similar approach of SCYM method developed by Lobell et al. (2015), applying it to winter wheat cultivation in central Spain, in the regions of Madrid, Guadalajara, Toledo and Cuenca. However, it introduces several key differences and innovations.

- Synthetic data generation was run for a total of 274,689 simulations using crop simulation model Daisy, with meteorological input spanning from 1981 to 2018, producing a wide selection of crop related variables.
- An extensive feature selection procedure was conducted to identify the most relevant explanatory variables for yield estimation.
- Instead of empirically converting the biophysical traits simulated by the model into a vegetation index, we used directly those traits that could potentially be derived with Earth Observation data
- Yield prediction models were implemented using machine learning approaches, specifically Random Forest and Symbolic Regression algorithms.
- Validation remote sensing variables are obtained through a data fusion method to include water-related metrics, making use of Sentinel-2, Sentinel-3 and Landsat imagery to compute land surface temperature (LST) (Guzinski et al., 2023), which is later used to estimate evapotranspiration values through the use of two source energy model (TSEB)

2 Methodology

Constructing the semi-empirical model for yield estimation through remote sensing data comprehends several tools. First, synthetic data are generated using a crop simulation model such as Daisy, which simulates crop growth and yield together with geo-hydrological dynamics given crop type, soil properties, field management practices, and meteorological forcing over a given period (e.g. several years) as inputs.

The synthetically generated dataset is then used to train machine learning models, specifically Random Forest and Symbolic Regression, to obtain the yield prediction.

Once the model is trained from the synthetic simulations, it is applied using actual Earth Observation data and in situ measured yield. EO data were obtained in a data fusion approach (Guzinski et al., 2023) combining Corine Land Cover maps (Copernicus.eu) with optical Sentinel-2, Sentinel-3 and Landsat imagery. These datasets are processed using different algorithms to derive biophysical variables, including evapotranspiration estimates obtained through the Two-Source Energy Balance (TSEB) model. On the other hand, crop yield in situ measurements at field level were gathered thanks to the Spanish Ministry of Agriculture ESYRCE system (Ministerio de Agricultura).

2.1. Synthetic data generation (Daisy)

The dataset required to calibrate the yield estimation model needs to be expanded both in terms of number of variables and of samples in order to cover as much as environmental variability and management practices as possible. The primary aim is to have a wide range of values regarding the yield and explanatory variables with which to train the model on. Furthermore, the amount and variability should fit model's precision and flexibility.

With these premises, real world data is not suitable for the task, especially because of limitation in the amount of time series observations of in situ yield measurements and its spatio-temporal variability. To solve this problem, synthetic data generation with Daisy crop model (S. Hansen et al., 1990) is adopted.

Daisy is a physically based model that serves for mechanistic simulation of agricultural fields. It keeps tracks of how the presence of water, Nitrogen and carbon evolves in the compound soil-crop-atmosphere system. Daisy can be performed both in one dimension or in two dimensions, for simulations in fields that have a presence of tile

drains or subsoil irrigation. A simulation depends on high temporal resolution meteorological data, generally hourly or daily, including air temperature, precipitation, humidity, wind speed and solar radiation.

The main processes implemented by the model are:

- Surface processes, taking place whereas water or solutes enter the system through rainwater, deposition, surface irrigation or spraying, and may be stored in or get past certain layers, such as canopy.
- Water flows in soils, described through Richards's equation (Richards, 1931) when the soil enter soil's matrix. Soil water fluxes are calculated by Darcy's law (Mollerup et al., 2014).
- Uptake of water by roots, simulated based on the single root concept (Gardner, 1960), driven by gradient in soil water pressure potential between the soil and the root surface. Roots length and density are as well influential in the uptake.
- Evapotranspiration, where Penman-Monteith (PM) equation is adopted in this case, retrieving potential evapotranspiration (ET_0) and actual evapotranspiration (ET) from relative soil water content (Allen, 2000). The method combines energy balance and mass transfer techniques. It first estimates the potential evapotranspiration (ET_0) through Equation 2, intended than to be multiplied to the crop coefficient (K_c), characteristic for each plant, and to a stress coefficient that accounts for water deficit (K_S) (Equation 3).
  The parameters and variables that are necessary to compute the potential evapotranspiration are the rate of change of saturation specific humidity with air temperature (Δ $[Pa \cdot K^{-1}]$), net radiation (R_n $[MJ \cdot m^{-2} \cdot day^{-1}]$), ground heat flux ($G$ $[MJ \cdot m^{-2} \cdot day^{-1}]$), air temperature at 2 meters height (T $[K]$), wind speed at 2 meters height (u_2 $[m \cdot s^{-1}]$), vapor pressure deficit (δe $[kPa]$) and psychrometric constant (γ $[\approx 66\ Pa \cdot K^{-1}]$)

$$ET_0 = \frac{0.408\Delta(R_n - G) + \gamma \frac{900}{T+273} u_2 (e_s - e_a)}{\Delta + \gamma(1 + 0.34u_2)}$$

Equation 2: Penman-Monteith (PM) equation

$$ET = K_S K_C ET_0$$

Equation 3: ET retrieval through potential value ET_0 (PM)

This approach to find evapotranspiration is widely employed for its simplicity of use and is the current method adopted by the Food and Agriculture Organization (FAO). Potential evapotranspiration is computed for an ideal reference grass crop, with unlimited water and nutrients resources and a height

of 12 cm. It represents a benchmark value, used then to obtain the real evapotranspiration multiplying it by crop and stress coefficient.

- Plant growth, through a model consisting of carbon flow and other processes, whereas the photosynthesis either follows a simple light response curve or a physiologically based model (DE PURY & FARQUHAR, 1997). The canopy structure is represented by a predefined vertical leaf density distribution, which varies with height as a function of the crop development stage and leaf area index (LAI). In turn, LAI is determined by leaf biomass and the development stage. Development Stage (DS) influences plant's partitioning in stem, leaf, dead material and storage organs.
- Soil Organic Matter (SOM), consisting of three pools, representing SOM with slow decomposition rate, fast decomposition rate and inert. The organic matter can be either in form of humus, made by microbial biomass or added through plant residues, organic fertilizers or compost.

In order to run Daisy several specifications are needed:

- Meteorological forcing, including air temperature, precipitation, humidity, wind speed and solar radiation
- Crop type
- Soil type and bulk density, in terms of soil composition (clay, silt, sand) and of soil's dry bulk density
- Humus concentration, for each soil horizon
- Management conditions, such as fertilizing quantity and irrigation

2.2. Random Forest and Symbolic Regression modelling

Another key component of the study is Random Forest and Symbolic Regression models. They are selected as machine learning techniques capable of finding complex relationships between the simulated variables about crop growth and crop yield itself. Both are data driven algorithms, which find a relationship between input and output with no a priori hypothesis about the system, but through pure data inference.

2.2.1. Random Forest

Random Forest belongs to non-parametric machine learning algorithms. In contrast to parametric machine learning methods, such as artificial neural networks (ANN), which aim to identify a single complex parametric function capable of representing the entire dataset, this approach is based on the construction of multiple local, non-parametric models by partitioning the input space into smaller regions and exploiting local patterns for output prediction [20].

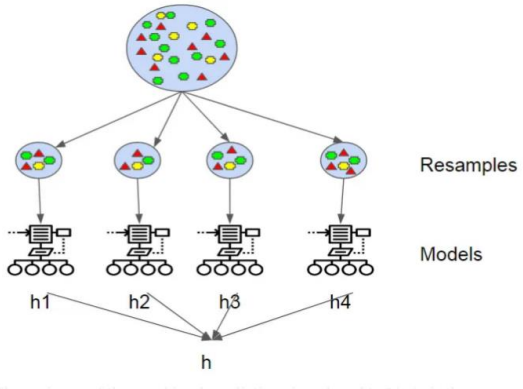


Figure 2: Random Forest schematic representation (Yagnik Pandya, n.d.)

Random Forest is an ensemble learning algorithm that combines multiple decision trees trained on different bootstrapped versions of the original dataset. Bootstrap consists of random sampling with replacement, generating multiple training sets D_b, on which deep Classification and Regression Trees (CART) are independently trained. A CART is a binary decision tree that recursively partitions the input space into increasingly homogeneous regions (Breiman et al., 2017). At each node, the split is defined by a single input variable and a threshold value, while predictions are assigned at the leaf nodes based on the training samples reaching them. Each individual tree typically exhibits low bias and high variance, while the final model prediction is obtained by averaging the outputs of all trees for regression problems.

In addition to bootstrap aggregation, Random Forest introduces further randomization by selecting a random subset of input variables at each split, increasing tree diversity and reducing overfitting (Breiman, 2001).

In this study, Random Forest regression was implemented using the "scikit-learn" Python library (Pedregosa & al., 2011). The model was trained on bootstrapped subsets of the training data, and final predictions were obtained by averaging the outputs of all trees.

2.2.2. Symbolic Regression

Symbolic Regression (SR) on the other hand is a machine learning approach that, unlike models such as Random Forest, does not build decision trees or "black box" predictions, but instead searches for interpretable mathematical equations describing the relationship between inputs and outputs (Makke & Chawla, 2024). SR explores the space of possible functions by combining mathematical operators (+, −, *, /, log, exp, etc.) and selecting those that best fit the data, without assuming a predefined functional form. This allows for interpretable models, expressed explicitly in a parametric mathematical formula, where the influence of each variable can be understood, in contrast to Random Forest, which is highly predictive but less

transparent. Moreover, SR typically produces a single global function summarizing system behaviour, whereas Random Forest relies on multiple local models aggregated together.

A software that provides a simple and user-friendly use of Symbolic Regression models is Eureqa (Data Robot), developed in Cornell's University Artificial Intelligence Lab (The Cornell University AI for Science Institute, CUAISci) and later commercialized by Nutonian Incorporation, which was acquired by Datarobot in 2017. The modelling engine makes use of genetic algorithms to determine the mathematical equations of Symbolic Regressions models. In this study, a 30 day free trial version of Eureqa was utilized.

2.3. Earth Observation data

The validation procedure of the calibrated models is performed on a set of variables, such as crop growth and evapotranspiration, the latter derived from remote sensing Land Surface Temperature (LST) data, through two source energy balance (TSEB) method (Norman et al., 1995). LST data are not currently obtainable through any of the existing spaceborne thermal sensors, at least at a quasi-daily field scale resolution. Thermal data sharpening serves to overcome this issue, with a combination of shortwave-multispectral Sentinel-2 observations, thermal-infrared Sentinel-3 observations and Landsat LST. This technique, developed by Guzinski et al. (2023), permits to improve data fusion methods, already applied for computation of evapotranspiration, that combines Sentinel-2 shortwave observations data, having a high spatial resolution of 10-20 m but a low temporal resolution of 5-day geometric revisit time at the equator, with Sentinel-3 LST with a daily temporal resolution but a 1 km spatial resolution.

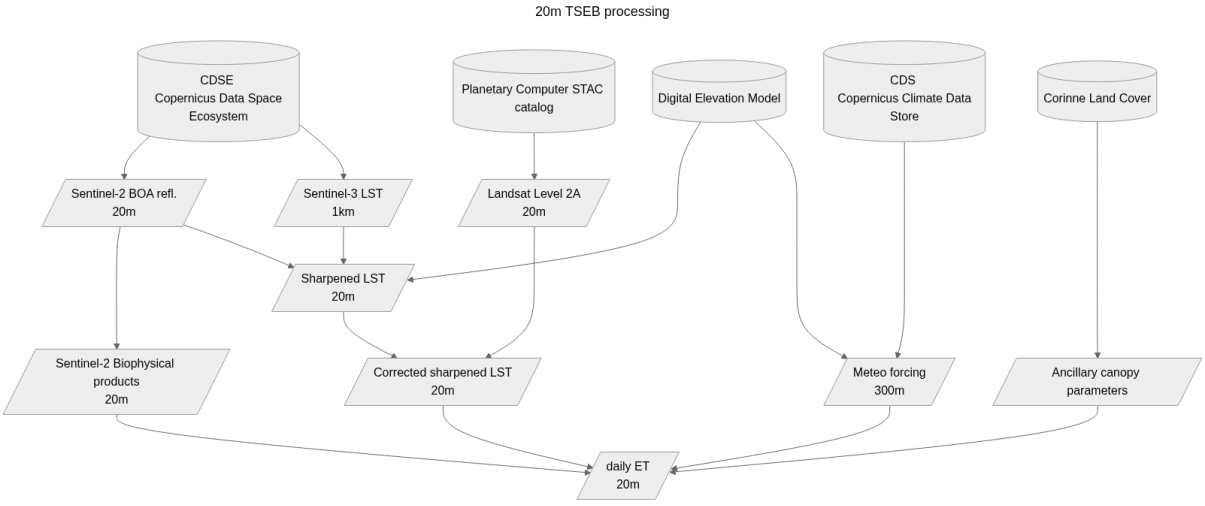


Figure 3: Flowchart of EO processing

In fact, the cited S2-S3 fusion method was exposed to be having issues of detecting short-term wetting events and of reproducing very low LST values, typical of irrigated fields, leading to ET underestimation from those fields (Guzinski et al., 2023).

Including a high spatial resolution LST, as Landsat LST 30 m product (Landsat Collection 2 Surface Temperature, United States Geological Survey), allows an enhancement in reproducing low values present in irrigated fields (Guzinski et al., 2023).

2.4. Two Source Energy Balance model (TSEB)

Evapotranspiration is one of the key variables in crop growth systems, representing the second-largest water flux after precipitation. (Arnold et al., 1999). It describes the water consumption in the coupled soil-crop systems, considering the contribution of humidity both leaving the soil (evaporation) and being released from leaves stomata (transpiration).

In literature, traditionally, the most popular method for estimating evapotranspiration is through the previously mentioned Penman-Monteith (PM) Equation 2. This approach provides a single aggregated estimate that implicitly accounts for the main processes involved in evapotranspiration, computing a reference, theoretical value of ET, using a set of standardized parameters. This is a substantial approximation, that can introduce several sources of uncertainty.

- The use of a crop coefficient (K_c), that has to be properly calibrated for each crop
- Does not consider explicitly the structure of the crop (height, leaf area index (LAI))
- Does not integrate radiometric observations, and so it does not have strong sensibility on vegetation actual water stress

For these reasons, other methods have been developed for estimating the evapotranspiration, such as the two source energy balance (TSEB) model. The approach is based on thermal remote sensing to separately estimate the two components of evapotranspiration, soil evaporation and plant transpiration (Figure 4) (Norman et al., 1995). This partitioning is performed through the surface energy balance, in which net radiation is split between the soil and canopy components.

$$R_n = R_{ns} + R_{nc} = H + \leq + G$$

Equation 4: total net radiation partitioning in soil and canopy components

$$R_{ns} = H_s + \mathrm{LE}_s + G$$

Equation 5: soil net radiation components

$$R_{nc}=H_c+\mathrm{LE}_c$$

Equation 6: canopy net radiation components

Net radiation (R_n $[W\ m^{-2}]$) is given by the sum of soil (R_{ns} $[W\ m^{-2}]$) and of canopy (R_{nc} $[W\ m^{-2}]$) contributions. The total component consists of the sum of sensible heat flux (H $[W\ m^{-2}]$), latent heat flux ($[W\ m^{-2}]$) and soil heat flux (G $[W\ m^{-2}]$). These variables represent the main heat fluxes governing the surface energy balance: the sensible heat flux (H), associated with temperature differences between the surface and the atmosphere; the latent heat flux (LE), related to phase changes of water through evaporation and transpiration; and the soil heat flux (G), which accounts for the energy stored in or released from the soil.

Due to its physical meaning, G is only considered in Equation 5 relative to soil and is estimated through an empirical relationship with the net radiation at the soil surface:

$$G=c_G R_{ns} \text{ with } c_G=0.35$$

Equation 7: empirical relationship to compute G (Kustas & Norman, 1999)

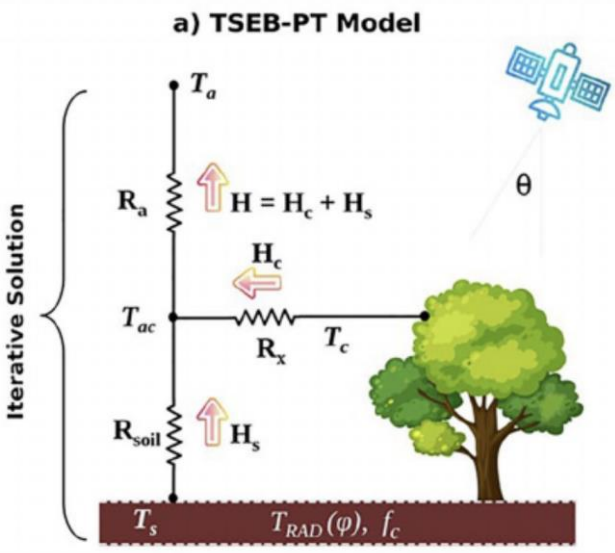


Figure 4: Schematic representation of two source energy balance - Priestley-Taylor model (Jaafar et al., 2022)

In the TSEB–PT formulation, the observed land surface temperature ($T_{RAD}(\theta)$ $[K]$) is partitioned into soil (T_s $[K]$) and canopy (T_c $[K]$) temperatures using an LAI-derived vegetation gap fraction that accounts for the sensor viewing geometry, through the nadir view angle of the thermal sensor (θ) (Kustas & Norman, 1999). These component temperatures are then used to estimate the sensible heat fluxes from the soil and

canopy through a resistance network (Figure 4), allowing the total sensible heat flux to be computed.

$$T_{RAD}(\theta) \approx [f(\theta)T_c^4+(1\text{-}f(\theta)T_s^4]^{0.25}$$

Equation 8: LST partitioning in soil and canopy temperature

The latent heat flux is subsequently derived from the surface energy balance. Since a priory we have two unknowns (Tc and Ts) and only Equation 8, an iterative approach is performed in TSEB. Canopy transpiration is initially assumed to occur at its potential rate following the Priestley–Taylor formulation (Equation 8) (Norman et al., 1995), while soil evaporation is obtained as the residual of the energy balance (Equation 4).

$$\mathrm{LE}_c=\alpha_{PT} f_G \frac{sp}{sp+\gamma} R_{nc}$$

Equation 9: Priestely-Taylor formula to initially find canopy transpiration

Where: α_{PT} and f_G are dimensionless Priestley Taylor parameter and fraction of LAI that is green and sp and γ are the slope of saturation vapor pressure versus temperature curve and the psychrometric constant, both in $[\frac{kPa}{C}]$.

With this initial estimate of LE_c, canopy and soil temperatures can be estimated (the latter by inversion of Equation 8) and thus the partitioning of fluxes between soil and canopy. However, at this stage LE_s might yield negative values indicating soil condensation that are not expected during daytime. Therefore, the iteration in TSEB is activated by sequentially reducing the α_{PT} coefficient and recomputing fluxes and temperatures until realistic fluxes (i.e. $\mathrm{LE}_s \geq 0$ and $\mathrm{LE}_c \geq 0$) are retrieved.

TSEB-PT, compared to applying FAO56 Penman Monteith equation to retrieve ET, allows the use of direct measurements of the radiometric temperature and biophysical traits, an estimation consistent with actual water status and growth conditions of the crop and the application of a less parametric dependent method.

3 Model design

This chapter describes the practical implementation of the workflow introduced in the previous sections, detailing how synthetic data generation, model training, and validation were carried out, focusing on the operational steps followed to build the yield estimation model.

3.1. Daisy

The first step of the model implementation consists in generating synthetic crop and yield data using the Daisy crop simulation model. The simulation workflow adopted in this study relies on an existing modelling pipeline, which was adapted to the study area and crop conditions considered in this work.

The main modifications introduced concern the selection of meteorological inputs, the configuration of winter wheat management practices, including irrigation conditions, and the definition of parameter ranges used to generate multiple simulation scenarios.

The workflow is organized into three main components:

1. retrieval and preparation of meteorological forcing data;
2. definition of the agricultural field configuration through a simulation template;
3. automated execution of large numbers of simulations through parameter sampling.

3.1.1. Meteorological data retrieval

Meteorological forcing data used in Daisy simulations were obtained from the ERA5 reanalysis dataset through the DestinE data platform (Earth Data Hub, Destination EU). Data retrieval and conversion into Daisy weather format were performed using the Python script get_meteo_destine.py.

The meteorological dataset used in the simulations is defined as follows:

- data source: hourly ERA5 reanalysis dataset accessed via the DestinE platform [1]
- study area: central Spain coordinates
- temporal coverage: 1981–2020

- processing: downloaded ERA5 variables are converted into a Daisy compatible weather file (.dwf) used as meteorological forcing for all simulations.

3.1.2. Daisy template

The Daisy simulations are defined through a configuration template ("daisy_simulations_template.dai"), which specifies the field setup for a one-dimensional (1D) soil–plant–atmosphere system. For each simulation run, predefined placeholders in the template are automatically replaced with the corresponding input parameters. The template is designed to:

1. read the meteorological forcing from the Daisy weather file generated in the previous step;
2. define a set of input parameters that are automatically assigned at each simulation run, namely:
    - *CLAY*, soil clay fraction [-];
    - *SILT*, soil silt fraction [-];
    - *SAND*, soil sand fraction [-];
    - *BULK_DENSITY*, dry soil bulk density, defined as the mass of dry soil per unit volume [$g \cdot cm^{-3}$];
    - *HUMUS_A*, humus content in soil horizon A [%];
    - *HUMUS_B*, humus content in soil horizon B [%];
    - *C_PER_N*, ratio between carbon and Nitrogen content in soil organic matter [g_C/g_N];
    - *N_PRECISION*, fertilization parameter defining the amount of mineral fertilizer (NPK02) applied per fertilization event per unit area [kg_N/ha];
    - *W_PRECISION*, parameter controlling the temporal spacing between irrigation events in the simulation [days];
3. define the soil column used in the simulations through the following components:
    - soil profile composed of three horizons with fixed thicknesses, namely horizon A (0–30 cm), horizon B (30–100 cm), and horizon C (100–400 cm) (Hiederer, 2009), where soil texture and bulk density remain constant along the column while humus content varies between layers, with sampled values applied to horizons A and B and a fixed humus content of 10% assigned to horizon C;
    - groundwater conditions defined with a deep-water table configuration, preventing direct groundwater influence on root-zone water dynamics;

- evapotranspiration computed using the Penman–Monteith formulation implemented in Daisy;

4. define crop-specific field management actions, which in this study are configured for winter wheat cultivation and include:
    - irrigation management, where irrigation events are triggered only when soil water content drops below field capacity (−33 kPa) (MARKOVIĆ et al., 2015) at a depth of 10 cm; each irrigation event applies water at a rate of 5 mm h^{-1} for one hour, and subsequent irrigation events can occur only after at least W_PRECISION days have passed and the soil moisture depletion threshold is again exceeded;
    - agricultural field management operations, consisting of the sequence of actions defined for winter wheat cultivation, including fertilization events, seedbed preparation, sowing, crop development monitoring, repeated irrigation activities, and harvesting operations, as detailed in the following pseudo-algorithm [source for the template, or in general for winter wheat growing dates in Spain].

Algorithm 1 Winter wheat activity definition

```
wait until September 1
wait until soil is trafficable or October 15 is reached
apply mineral fertilizer NPK02 at rate N_PRECISION [kg N/ha]
perform seed bed preparation
sow winter wheat crop
wait until crop development stage reaches 0.2 (short after emergence)
apply mineral fertilizer NPK02 at rate N_PRECISION [kg N/ha]
repeat irrigation events according to irrigation rules
wait until crop development stage reaches 1.9 (near grain maturity)
harvest winter wheat leaving an 8 cm stub
```

The crop development stage (DS) is used to describe the phenological progress of the crop, where DS = −1.0 corresponds to sowing, DS = 0.0 to crop emergence, DS = 1.0 to flowering, and DS = 2.0 to full maturity or ripening (S. Hansen et al., 1990). Consequently, allowing the crop to reach DS values close to 2.0 (e.g., 1.9) ensures that harvest occurs when the crop is close to physiological maturity.

3.1.3. Daisy simulation

The automated execution of Daisy simulations is performed through the script "*get_daisy_simulations.py*", which controls the generation of multiple crop simulation scenarios and the execution of Daisy runs.

The script combines the simulation template described in the previous section with the meteorological files generated through the meteorological retrieval procedure, producing a large ensemble of simulations corresponding to different soil and management conditions.

Simulation parameters are sampled within predefined ranges using a quasi-Monte Carlo approach based on Sobol sequences (Sobol', 1967) generated through the Saltelli sampling scheme implemented in the SALib library (Herman & Usher, 2017). For each sampled parameter combination, the placeholders in the Daisy template are replaced with the corresponding random values, and a simulation is executed using the selected meteorological year.

The workflow implemented in the script performs the following operations:

- selection of the crop type to be simulated, which in this study is winter wheat;
- loading of meteorological data generated through the meteorological preprocessing script;
- sampling of soil and management parameters within predefined ranges;
- automatic generation of configuration files for Daisy simulations;
- execution of Daisy runs and storage of the resulting crop and yield outputs.

A total of 12000 parameter combinations were generated. Due to simulation failures occurring for some meteorological years, mainly associated with the crop not reaching the target development stage required for harvest, simulations were successfully completed for 23 years instead of the full meteorological record. After excluding failed runs, the final dataset consists of 274689 successful crop simulations, which constitute the synthetic dataset used for model training.

The parameter ranges used for the sampling procedure are summarized in **Error! Reference source not found.**, which reports the minimum and maximum values assigned to each variable varied during the simulation process.

Soil texture fractions are parameterized such that clay is expressed as an absolute fraction of total soil composition, while sand is defined as a fraction of the remaining non-clay component. The silt fraction is then computed as the remaining portion needed to complete the soil texture composition.

Parameter	Min	Max	Unit
clay	0.04	0.45	[-]
sand	0.26	0.94	[-]
bulk_density	0.926	1.527	$g \cdot cm^{-3}$
humus_a	0.5	3.0	%
humus_b	0.05	0.9	%
c_per_n	6.8	17.2	g_C/g_N
n_precision	0	50	kg_N/ha
w_precision	1	29	days

Table 1: range of variability selected for each parameter

The parameter ranges in **Error! Reference source not found.** represent plausible soil and management conditions in the study area while maintaining sufficient variability to generate diverse simulation scenarios for model training.

In particular, "*w_precision*", controlling the minimum interval between irrigation events, was varied to account for differences in water availability and irrigation practices among farmers in the study region.

3.2. Data cleaning and exploratory analysis

Before calibrating the yield prediction models, the synthetic dataset generated through Daisy simulations was processed in order to remove inconsistencies, construct aggregated variables, and identify explanatory variables suitable for model training. This step is necessary because the simulation output contains variables that are redundant, not observable in real operational conditions and quantities strongly correlated or constant across simulations.

The data cleaning and exploratory analysis phase consists of completing these tasks while conserving the variability required for robust machine learning training and to analyse the statistical behaviour of the simulated variables, to better understand their relationships and variability.

3.2.1. Dataset preprocessing and feature preparation

The dataset used for model calibration consists of the outputs generated by Daisy simulations, stored as tabular data containing 226 crop, soil, and environmental variables together with the simulated yield values. A series of columns were initially removed from the raw dataset.

- Simulation setup variables, including the meteorological year considered, the harvest date (month and day) and the soil and field management parameters defined in the Daisy simulation template.
- Variables corresponding to crop component values at harvest time were removed, since in operational remote sensing applications it is difficult to retrieve information exactly at harvest date due to possible cloud cover (Guzinski et al., 2023) and satellite overpass frequency.
- variables describing individual plant components were excluded, since from satellite observations measurements it is not possible to retrieve measurements of separate plant organs (Lobell et al., 2015).
- Plant Area Index (PAI) related variables were removed, as they are linked with LAI through a linear relationship.
- Storage organs dry matter at harvest time, representing crop yield [tons/ ha], was extracted from the simulation outputs and used as the target variable for model calibration.

The available explanatory variables are:

1. Leaf Area Index (LAI) [m^2/m^2]. For these variable, seasonal statistics are available, including maximum, mean and standard deviation values, the day of year at which selected fractions of its maximum values are reached and monthly mean values;
2. total plant dry matter (WPlant, [tons/ha]), equivalent to above ground dry biomass, and plant Nitrogen content (NPlant, [kg_N/ha]). As for canopy variables, seasonal statistics, threshold attainment days and monthly mean values are available;
3. reference evapotranspiration (ET_ref), potential evapotranspiration (ET_0), potential transpiration (T_0), actual evapotranspiration (ET) and actual transpiration (T), all expressed as monthly accumulated values [mm/month];
4. monthly net photosynthesis (NP), expressed as [g_{CO2}/m^2/month];

After importing the dataset and removing the previously mentioned sets, data were randomly shuffled to avoid ordering effects related to simulation batches.

Then, variables showing constant values across simulations are removed, since they do not contribute for model calibration; these variables correspond to monthly quantities computed during periods in which the crop was not yet present or already harvested, resulting in null values for all available simulations.

The resulting dataset is therefore reduced to a consistent set of candidate explanatory variables, which are analysed through statistical and correlation based methods. The

following dataset is referred to as *potential dataset*, since it is formed by all the available features that are potentially retrievable from remote sensing.

A second dataset was built, consisting of the variables that have a correspondent in the validation series. This set is named *operational dataset*, since it is the one against with the best model is tested through real world data. To obtain it:

- NPlant, NP and ET_ref related variables were removed, since there are no associated quantities in validations data;
- features describing the day of year of maximum development and the days required to reach fractions equal to 20% and 80% of maximum LAI and plant biomass were excluded, since there are no associated quantities in validations data.

***Potential* dataset**		***Operational* dataset**	
Quantity	**Associated variables**	**Quantity**	**Associated variables**
LAI WPlant NPlant	mean_*var*, max_ *var*, std_ *var*, doy_max_ *var*	LAI WPlant -	mean_*var*, max_ *var*, std_ *var*
	days_0.1_ *var*, days_0.2_ *var*, days_0.5_ *var*, days_0.8_ *var*, days_0.9_ *var*,		days_0.1_ *var*, days_0.5_ *var*, days_0.9_ *var*,
	month_01_ *var*, month_02_ *var*, month_03_ *var*, month_04_ *var*, month_05_ *var*, month_06_ *var*, month_07_ *var*, month_10_ *var*, month_11_ *var*, month_12_ *var*		month_01_ *var*, month_02_ *var*, month_03_ *var*, month_04_ *var*, month_05_ *var*, month_06_ *var*, month_07_ *var*, month_10_ *var*, month_11_ *var*, month_12_ *var*

ET_ref ET_0 T_0 ET	month_01_ *var*, month_02_ *var*, month_03_ *var*, month_04_ *var*, month_05_ *var*, month_06_ *var*, month_07_ *var*, month_08_ *var*, month_09_ *var*, month_10_ *var*, month_11_ *var*, month_12_ *var*	- ET_0 T_0 ET	month_01_ *var*, month_02_ *var*, month_03_ *var*, month_04_ *var*, month_05_ *var*, month_06_ *var*, month_07_ *var*, month_08_ *var*, month_09_ *var*, month_10_ *var*, month_11_ *var*, month_12_ *var*
T NP	month_01_ *var*, month_02_ *var*, month_03_ *var*, month_04_ *var*, month_05_ *var*, month_06_ *var*, month_07_ *var*, month_10_ *var*, month_11_ *var*, month_12_ *var*	T -	month_01_ *var*, month_02_ *var*, month_03_ *var*, month_04_ *var*, month_05_ *var*, month_06_ *var*, month_07_ *var*, month_10_ *var*, month_11_ *var*, month_12_ *var*

Table 2: *Potential* and *operational* dataset; *var* indicates that all the quantities presented in the same row in the quantity column have a corresponding variable.

3.2.2. Descriptive statistics

Before performing correlation analysis and model calibration, the statistical properties of the variables were examined to understand better their distribution and variability across simulations. Basic descriptive statistics such as mean, standard deviation and Coefficient of Variation (CV, fraction given by the division of the standard deviation over the mean) were computed for each variable and for the target yield Also skewness

and Fisher's kurtosis indicators were found to evaluate distribution asymmetry and tail behaviour. Outliers were also identified, as records whose standardized values fell outside of the range defined by plus or minus three standard deviations around the mean. The quantitative analysis was supported by qualitative inspection through boxplots and histograms of the most extreme variables.

3.2.3. Correlation analysis

Following the descriptive statistical analysis, correlation analysis was performed to identify variables most strongly related to crop yield and to evaluate redundancy among explanatory variables. Before computing correlations, the dataset was divided into training and testing subsets, using 80% of the samples for model calibration and the remaining 20% for performance evaluation.

Pearson and Spearman correlation coefficients were computed between each candidate explanatory variable and target yield variable. Pearson correlation is used to quantify linear relationships, while Spearman correlation captured monotonic relationships that may not be strictly linear. To guarantee statistical significance, variables with p-values greater than 0.01 were discarded from further analysis.

Variables were then ranked according to a score defined as the maximum absolute value between their Pearson and Spearman correlations with crop yield and the 50 variables showing the strongest associations were selected as candidate predictors for further analysis. Cross correlations between these top 50 variables were examined, using Pearson correlation, to identify predictors describing similar physical processes or crop conditions. This analysis provides a qualitative overview of redundancy within the feature set.

The results were then used to support the selection of compact sets of candidate predictors employed in the following modelling steps.

3.2.4. Redundancy reduction and representative variable selection

To further reduce redundancy among candidate predictors, a hierarchical clustering approach based on inter variable correlations was applied. This method groups variables according to similarity in their behaviour across simulations, allowing predictors describing comparable crop or environmental processes to be identified and organized into clusters. The clustering was performed using distances derived from Pearson cross correlations among variables.

The clustering structure was visualized through a dendrogram, which supports the identification of groups of highly correlated variables. From each cluster, a representative variable, which is the one with the highest Pearson correlation coefficient with the output, was selected in order to retain the main information content while reducing the dimensionality of the predictor set used for model calibration.

Based on this clustering structure, compact predictor subsets of different sizes were constructed, namely:

- 2 variables;
- 4 variables;
- 8 variables;
- 12 variables.

These subsets were defined to evaluate model performance under progressively increasing levels of predictor complexity, balancing predictive accuracy with model simplicity and practical applicability, since models based on fewer variables are easier to interpret and require fewer input data to be collected in operational agricultural contexts.

This procedure results in a more compact and interpretable set of explanatory variables, limiting multicollinearity effects and improving the robustness of subsequent machine learning models.

3.3. Random Forest

Random Forest models were implemented in Python using the "*run_random_forest*" dedicated function available in the scikit-learn library (Pedregosa & al., 2011), which performs model training and model fitting on the training dataset and computes predictions on the testing subset.

Model performance was evaluated using the coefficient of determination (R^2) and the root mean square error (RMSE) computed on test data. In addition, feature importance scores provided by the Random Forest algorithm were extracted to quantify the relative contribution of each predictor to yield estimation.

The main hyperparameters used in the model configuration include:

- number of trees in the ensemble (*n_estimators* = 500), chosen to ensure stable predictions, but for low computational costs;
- unrestricted tree depth (max_depth = None), allowing trees to fully adapt to data structure;
- minimum number of samples required in terminal nodes (min_samples_leaf = 2), to avoid single observations splits and thus to limit excessive overfitting;
- fixed random seed for reproducibility of the results;
- parallel computation enabled across available CPU cores.

The trained model object is also retained for subsequent validation steps.

To evaluate the performance of the Random Forest model under different levels of input complexity, models were trained using previously mentioned subsets of 2, 4, 8, and 12 variables derived from the *potential* dataset. This allows assessment of how model accuracy varies with the number of predictors when only using synthetic data generated by Daisy simulations.

Subsequently, to assess the applicability of the approach in real-world conditions, the model was also trained on the *operational* dataset and tested using real world validation data, considering the configuration with 12 variables only.

For each configuration, a simple linear regression model was used as a baseline benchmark.

3.4. Symbolic Regression

The Symbolic Regression model was developed using Eureqa (Data Robot), employing as input the 50 most correlated variables from the *operational* dataset. This choice allows the algorithm to explore a wide set of candidate predictors. Unlike Random Forest, which will use all the predictors, Symbolic Regression generates an explicit mathematical expression that involves only a limited subset of the provided variables. This results in a more interpretable formulation.

Since Symbolic Regression builds analytical formulas directly, prior to model construction, all input variables were to prevent differences in scale from influencing the search process.

The set of admissible mathematical operators to build the formulas were defined within the software. The selected operators included constants, input variables, basic arithmetic operations (addition, subtraction, multiplication, division), and non-linear functions such as sine, cosine, exponential, natural logarithm, power, and square root. This configuration enables the exploration of both linear and non-linear relationships while maintaining control over the structural complexity of the resulting equation.

After uploading the data and defining operators Eureqa searches for mathematical formulas that describe the output. The software returns a set of formulas, each with an characterized by a "size" and a "fit" metric. "Size" indicates the complexity of the formula, while "fit" quantifies its predictive performance.

Since the primary objective function used by the software is the absolute error, the reported "fit" corresponds to the normalized mean absolute error (MAE). The normalization is performed relative to a constant baseline model, where the output is set equal to its mean. Therefore, a "fit" value of 1 corresponds to the constant predictor, whereas lower values indicate improved predictive performance.

Model	*Potential* dataset	*Operational* dataset
Linear Regression (Baseline)	2, 4, 8, 12 variables	12 variables
Random Forest	2, 4, 8, 12 variables	12 variables
Symbolic Regression	-	Top 50 variables

Table 3: Models analyzed and input configuration

3.5. Validation data

The observational dataset used for model validation refers to agricultural plots located in central Spain, specifically within the provinces of Madrid, Guadalajara, Toledo, and Cuenca. The analysis covers agronomic years 2018, 2019, 2020, and 2021. An agronomic year is defined as the period from October the previous year to September of the year considered. For example, agronomic year 2018 includes data from October 2017 to September 2018.

Yield observations data were provided as fresh weight measures. In order to compare them with Daisy's yield output (storage organs dry matter at harvest time), a moisture correction factor of 14% was applied to the measurements, as commonly adopted in the literature (Strong & Sbur, 1960).

$$Yield_{DM,\,obs}=Yield_{FW,\,obs}\cdot(1\text{-}0.14)$$

Equation 10: Dry Matter (DM) observed weight computation from measured Fresh Weight (FW)

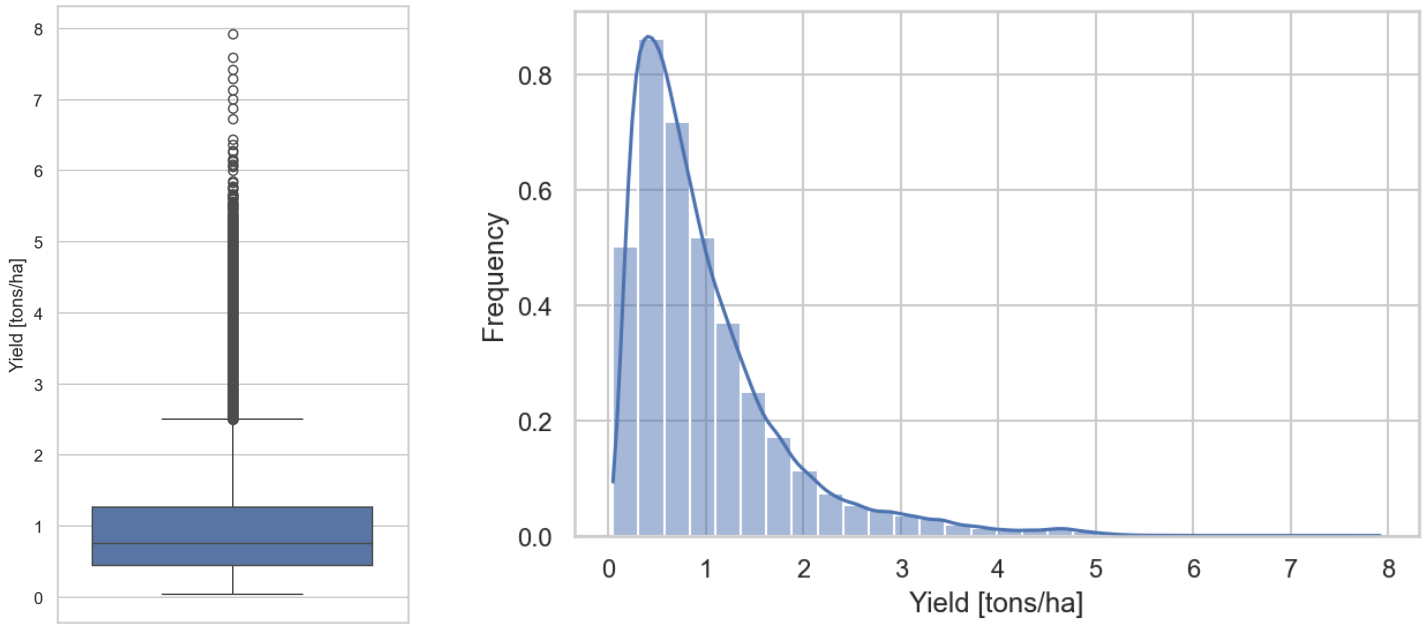
The final observational dataset was obtained by merging two datasets, one containing monthly variables and another containing annual variables, using agronomic year and a unique field identification code as matching keys. This operation resulted in a dataset of 982 observations, that was further reduced removing records that containing missing values, leading to a final table consists of 968 observations.

4 Results

In the following chapter the main results are reported, including a statistical analysis of the datasets used, the passages involved in the selection of the main explanatory variables and the performance of the models used to predict yield.

4.1. Descriptive analysis

A preliminary analysis of the data was done examining at exploratory variables and outputs basic statistics.



(a) Simulated yield boxplot.

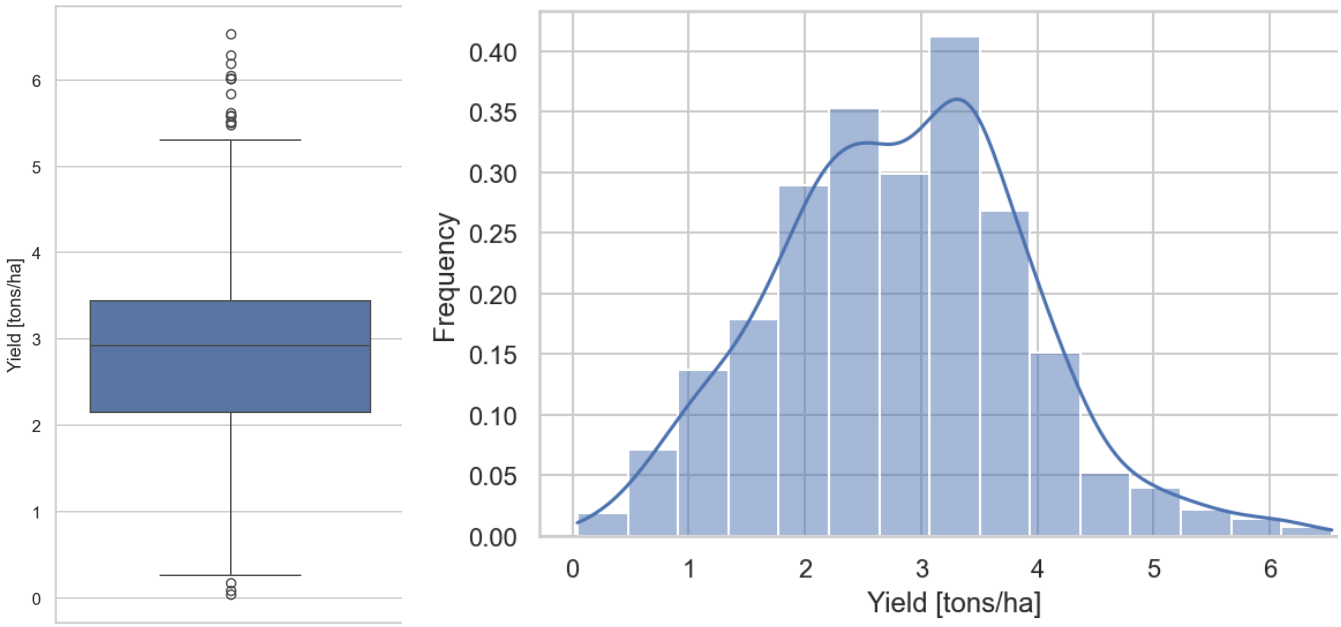
(b) Simulated yield histogram and interpolated distribution curve.

Figure 5: Simulated yield distribution graphs.

Simulated yield exhibits a mean of 0.90 tons/ha, a standard deviation of 0.80 tons/ha, and a maximum value of 7.92 tons/ha. The distribution is highly concentrated around the mean and qualitatively resembles a lognormal shape. This is also supported also

by skewness and Fisher's kurtosis values, respectively of 1.96 and 4.93, typical of a leptokurtic, right tailed function.

On the other hand, observed yield shows a mean value of 2.83 tons/ha and a standard deviation of 1.09 tons/ha. The maximum observed yield is equal to 6.53 tons/ha. The distribution of the measurements seems to follow, qualitatively, a normal shape.

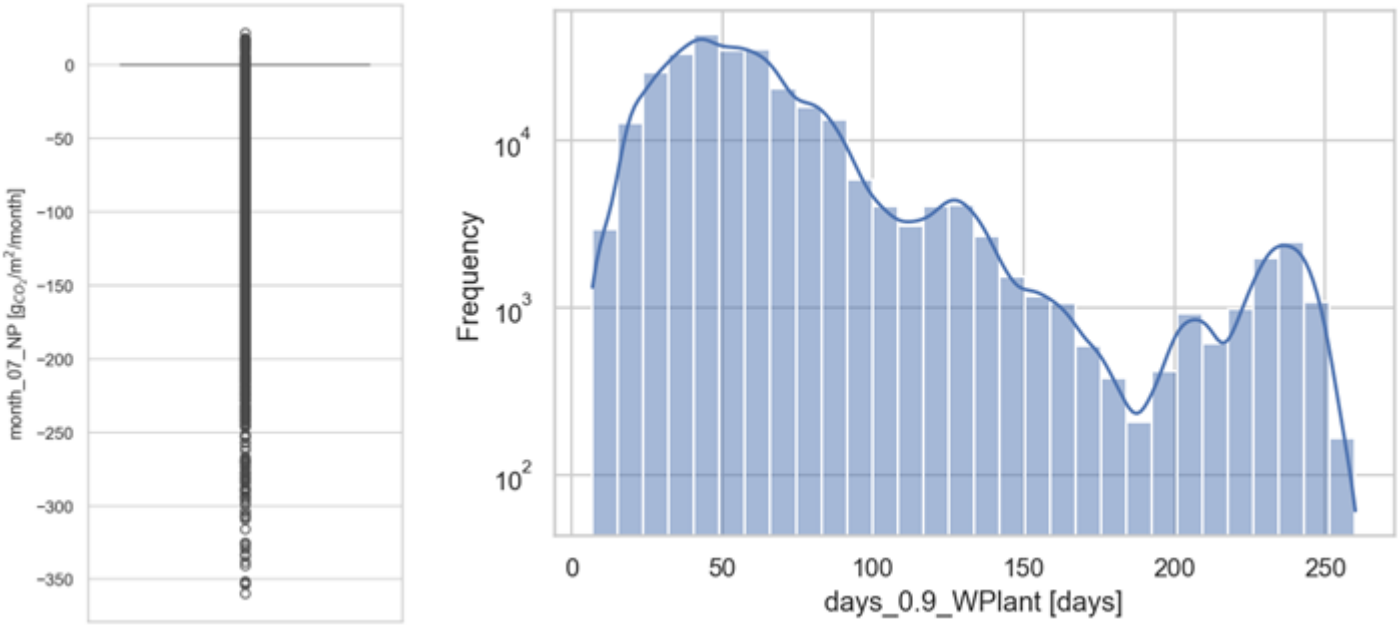


(a) Observed yield boxplot.

(b) Observed yield histogram and density curve

Figure 6: Observed yield distribution graphs

By taking into consideration the basic statistics of the candidate predictors with highest skewness values, especially the coefficient of variation (CV, equal to the standard deviation divided by the mean), it is possible to investigate some interesting insights into the simulation's values. *month_07* variables (number of simulations with harvest after June) were among the most the most skewed variables and with the highest CV. *month_07_NP* for example is null in 94% of the simulations. *days_0.9_WPlant* shows a peculiar distribution shape and is second in number of outliers (3.2%).



(a) *month_07_NP* boxplot.

(b) *days_0.9_WPlant* histogram and density curve; the vertical axis is in logarithmic scale

Figure 7: *month_07_NP* and *days_0.9_WPlant* distribution graphs

4.2. Correlation with output

From the correlation of the input variables to the target yield variable, it was possible to recognize the variables that explicate most of the outputs variance and to reduce the dimensionality of the dataset.

	Variable	Pearson	Spearman	Score
1	*month_05_NP*	0.666	0.568	0.666
2	*month_05_NPlant*	0.550	0.565	0.565
3	*max_NPlant*	0.536	0.560	0.560
4	*month_04_NP*	0.555	0.498	0.555
5	*month_03_NPlant*	0.523	0.549	0.549
6	*std_NPlant*	0.511	0.549	0.549
7	*month_04_NPlant*	0.526	0.545	0.545
8	*days_0.8_WPlant*	-0.428	-0.473	0.473
9	*max_WPlant*	0.398	0.458	0.458
10	*month_02_NPlant*	0.392	0.449	0.449

Table 4: top ten *potential* input variables with the highest correlation with the output

	Variable	Pearson	Spearman	Score
1	*max_WPlant*	0.398	0.458	0.458
2	*month_06_T*	0.321	0.448	0.448
3	*month_06_ET*	0.316	0.404	0.404
4	*month_05_WPlant*	0.324	0.403	0.403
5	*days_0.1_WPlant*	-0.365	-0.226	0.365
6	*days_0.9_WPlant*	-0.232	-0.360	0.360
7	*month_11_ET*	-0.357	-0.315	0.357
8	*month_12_ET*	-0.297	-0.353	0.353
9	*days_0.5_WPlant*	-0.350	-0.297	0.350
10	*days_0.9_LAI*	-0.325	-0.301	0.325

Table 5: top ten *operational* input variables with the highest correlation with the output

The comparison of the ten most correlated variables in the *potential* dataset and in the *operational* subset reveals that the seven top ranked variables in the *potential* dataset are not present in the *operational* one. These excluded variables are related to net photosynthesis or to Nitrogen content.

Negative correlation coefficients are observed for some of the evapotranspiration monthly variables, for months not close to the harvest, as well as for the *days_0.X_var* indicators. This suggests that within the simulation framework, prolonged periods above a certain fraction of the maximum value of a biophysical trait (e.g. plant maximum weight), lead to a lower value of the yield.

4.3. Cross correlation analysis

After reducing the dimensionality of the dataset by selecting the variables most correlated with the output, cross correlation analysis was performed to recognize groups of candidate predictors that were strongly correlated with one with each other. Figure 8 presents the correlogram based on the pairwise Pearson cross correlation among the 50 *potential* dataset variables most correlated with the output. The color scale spans from blue (low or no correlation), to green (moderate correlation) and yellow (near perfect correlation).

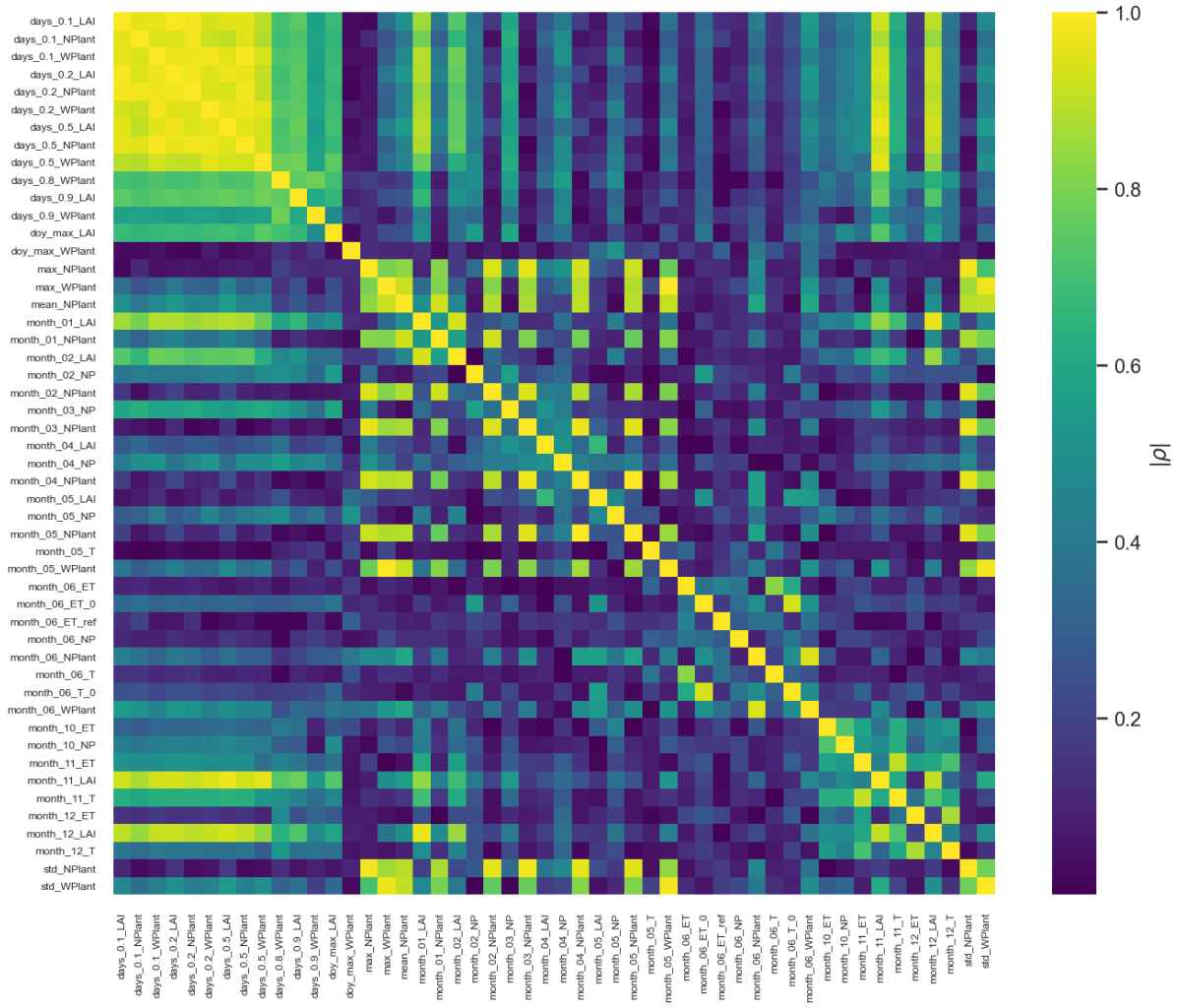


Figure 8: top 50 *potential* dataset variables Pearson correlation matrix heatmap

An inspection of the correlogram, proceeding from top rows downwards, reveals the following relationships:

- variables belonging to *days_0.X_var* family, and especially those with fractions equal to or below 0.5, are all strongly correlated one to another (top left yellow square in Figure 8). These variables also exhibit strong correlations with winter monthly averages (months 11, 12, 01 and 02);
- *days_0.9_WPlant* (i.e. number of days that are above the 90% WPlant percentile for this year) appears to be the least correlated with the others in the same group;
- Nitrogen and crop weight related yearly statistics variables and monthly values, for months up to May (month ≤ 05), shows strong mutual correlations (yellow and light green cells in the central square in Figure 8, ranging from *max_NPlant* to *month_05_WPlant*);

- variables related to June (month 06) variables are moderately correlated one to each other (square that goes form *month_06_ET* to *month_06_WPlant* in Figure 8)
- winter months variables show strong internal correlation (bottom right square, from *month_10_ET* to *month_12_T*)

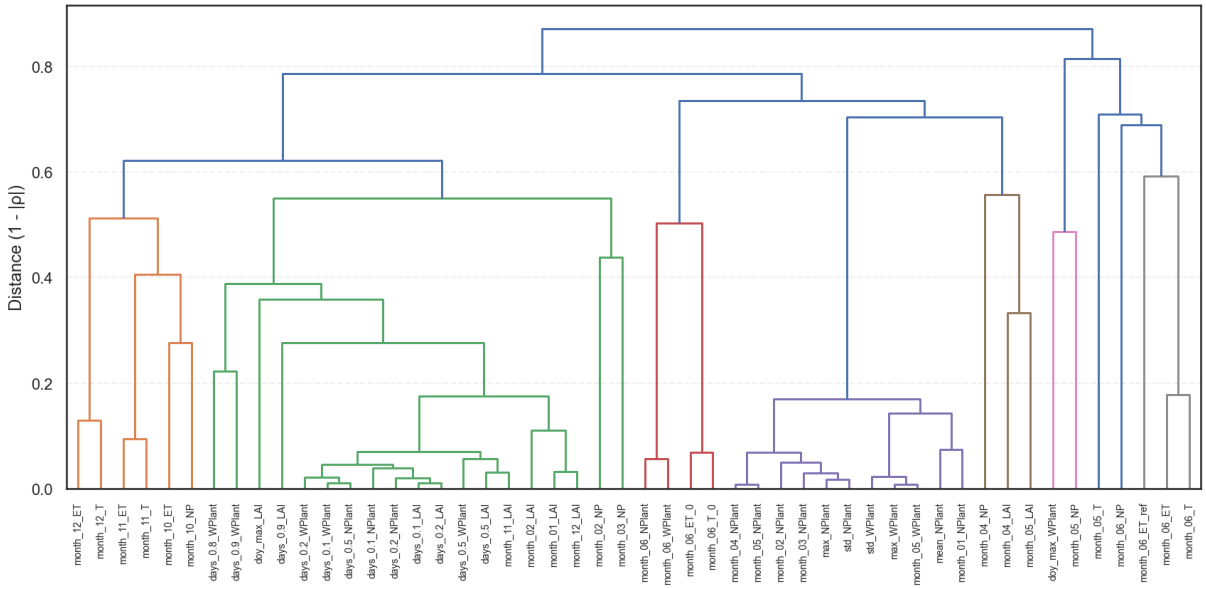


Figure 9: hierarchical clustering dendrogram based on correlation distance ($1 - |\rho|$) between top 50 *potential* variables

The dendrogram confirms the same patterns described in the correlogram, with winter variables clustering together, days variables forming a distinct cluster at a very low hierarchical distance, as well as Nitrogen related variables.

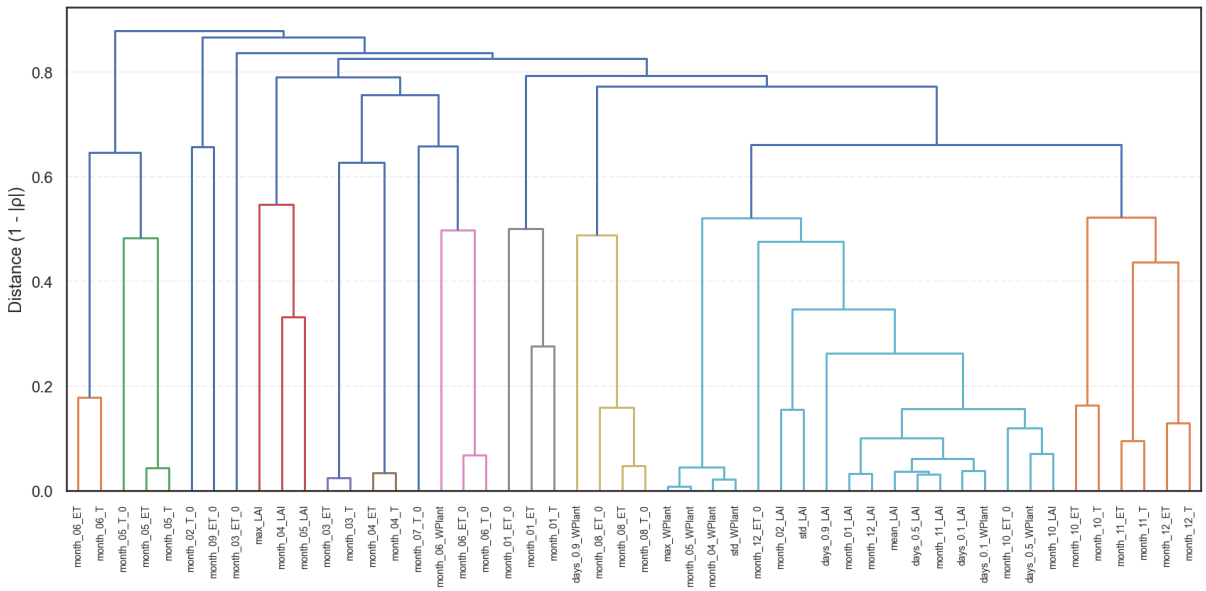


Figure 10: hierarchical clustering dendrogram based on correlation distance ($1 - |\rho|$) between top 50 *operational* variables

The dendrogram associated with *operational* (Figure 10) variables suggests that the variables are less correlated one to each other than in the *potential* dataset, since the clusters merge at a higher level of distance. May and June variables are visibly isolated from the rest of the group.

By cutting the dendrogram at different levels, four different groups of predictors were found for the *potential* dataset and one for the *operational* dataset.

Potential				***Operational***
Two predictors	**Four predictors**	**Eight predictors**	**Twelve predictors**	**Twelve predictors**
month_04_NP	*month_04_NP*	*month_04_NP*	*month_04_NP*	*month_06_T*
month_05_NP	*month_05_NP*	*month_05_NP*	*month_05_NP*	*month_02_T_0*
	days_0.8_WPlant	*days_0.8_WPlant*	*days_0.8_WPlant*	*month_09_ET_0*
	month_06_NP	*month_06_NP*	*month_06_NP*	*month_04_LAI*
		month_06_NPlant	*month_06_NPlant*	*month_03_T*
		month_05_NPlant	*month_05_NPlant*	*month_06_ET_0*
		month_06_T	*month_06_T*	*month_07_T_0*
		month_05_T	*month_05_T*	*month_01_ET*
			month_11_ET	*days_0.9_WPlant*
			month_02_NP	*max_WPlant*
			month_04_LAI	*month_11_ET*
			month_06_ET_ref	*month_03_ET_0*

Table 6: variable selections for different number of predictors

Considering *potential* dataset, Net Photosynthesis variables emerge as particularly relevant predictors of the output, given both their high correlation with the response variable and relatively low correlation with the other variables. The predictors in *potential* dataset that are also included in the *operational* one are: *month_06_T*, *month_04_LAI* and *month_11_ET*.

4.4. Models' performance with *potential* dataset

For each model's configuration trained on *potential* dataset, performance metrics could be obtained only relatively to their score in the selection of the model.

The following indicators were considered:

- R^2: measures the proportion of variance in the observed yield explained by the model. Values closer to 1 indicate a better fit, while values close to 0 indicate limited explanatory power;
- RMSE (Root Mean Squared Error): quantifies the average of the prediction error, in the same units as the target variable (tons/ha). Values closer to 0 indicate higher predictive accuracy;
- Pearson correlation coefficient between the observed and predicted yield.

A linear regression model was used for each set of variables, to confront the score of the models with a baseline benchmark.

Model	R^2	RMSE	Pearson
LR_2_*p*	0.555	0.532	0.745
RF_2_*p*	0.743	0.405	0.863
LR_4_*p*	0.816	0.342	0.904
RF_4_*p*	0.978	0.118	0.989
LR_8_*p*	0.902	0.249	0.950
RF_8_*p*	0.996	0.049	0.998
LR_12_*p*	0.914	0.234	0.956
RF_12_*p*	0.997	0.044	0.998

Table 7: performance metrics of *potential* variables trained models

Random Forest models outperform the linear regression for every configuration of variables. The RMSE values shows that the difference between the two grows as the model's complexity increases.

The feature importance provided by the Random Forest model was analyzed to further assess the relevance of each predictor. In regression trees, variable importance is computed as the total reduction in variance produced by splits involving a given variable, averaged over all trees in the ensemble and normalized, so that the sum

equals one. This metric reflects the relative contribution of each predictor to improving the model's predictive performance.

Results indicate that month_05_NP is the most influential variable, with an importance of approximately 0.40. The second and third most relevant predictors are month_04_NP and month_06_NP, with importance values of 0.18 and 0.12, respectively.

4.5. Models' performance with *operational* dataset

The models validated on real world observations were trained on *operational* dataset.

The Symbolic Regression model selected on Eureqa is the following:

$$Y=0.941+2.706 \cdot max\_WPlant+0.176*month\_06\_ET-0.141*month\_05\_LAI-2.455*std\_Wplant$$

Equation 11: Symbolic Regression mathematical model selected

This model was chosen since it had a good balance in complexity and performance, using only four predictors.

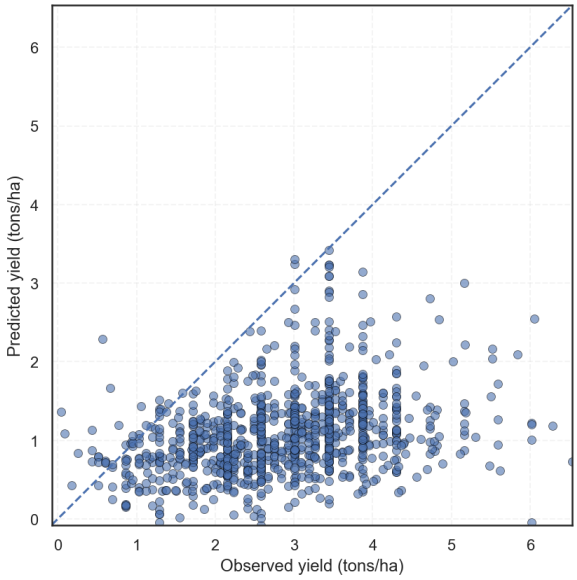
Regarding the importance of the variables in the Random Forest model, variable *days_0.9_WPlant* is the most influential variable, with a importance of 0.42, followed by *max_WPlant* with a value of 0.27 and *month_06_T* that has a value of importance of 0.11.

The performance of the model, together with the one of the Random Forest and of the linear regression, is shown in the following table:

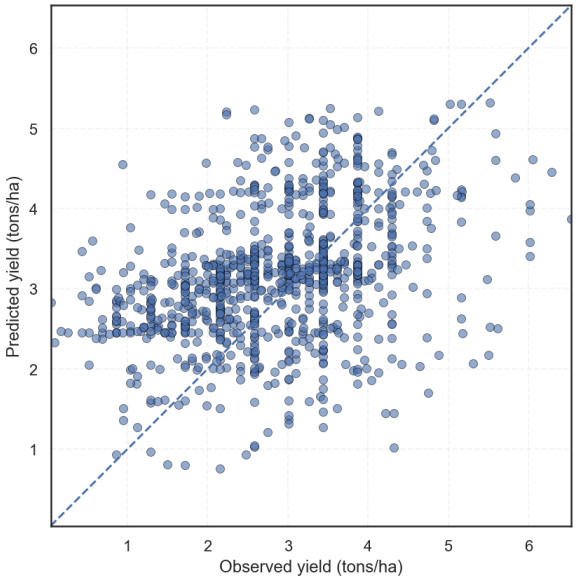
Model	R^2	RMSE	Pearson
LR_12_*o* (cal)	0.526	0.55	0.725
LR_12_*o* (val)	-2.597	2.041	0.331
RF_12_*o* (cal)	0.995	0.059	0.997
RF_12_*o* (val)	-0.094	1.126	0.374
SR_*o* (cal)	0.729	0.415	0.861
SR_*o* (val)	-0.118	1.147	0.269

Table 8: performance metrics of *operational* variables trained models and validated on observations data

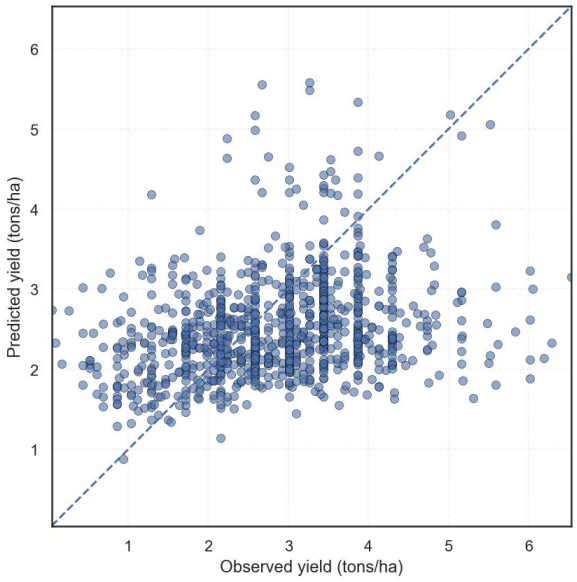
Comparing linear regression and Random Forest models performance in training on the *potential* dataset, there is a reduction for all the three metrics considered.



(a) Linear regression



(b) Random Forest



(a) Symbolic Regression

Figure 11: observed vs predicted yield scatter plot for linear regression, Random Forest models and Symbolic Regression models

Performance values drastically decrease when calibrated model is applied on real observations. Random Forest model outperforms the other two in every metric considered. Linear regression is subject to the most important decrease in R^2, while

regarding Pearson coefficient, linear regression estimated yield has a stronger correlation with the observed yield than Symbolic Regression. RMSE is about one ton per hectare for models based on machine learning approach, while the linear method returns a value of two tons.

The scatter plot for the three models confirms what was previously stated, with Random Forest model being able to predicts the yield the best, but with all three showing a poorer performance. Linear regression model, compared to the other two, displays a clear underestimation of the observed yield.

5 Discussion

The results shown in the previous chapter highlights a relatively poor performance for the used model in predicting the observed yield. The factors for this outcome are numerous, but it can mainly be explained by the statistical differences of the synthetically generated calibration dataset and the real-world observations used for validation.

5.1. Calibration and validation datasets

The dataset used for calibration of the model, result of the Daisy synthetic data generation, and for validation, where yield was provided by the Spanish Ministry of agriculture and remote sensing observation were retrieved as a processing of Earth Observations imagery, have substantial differences, described in paragraph 4.1.

5.1.1. Yield

Since the objective of the process was to obtain adaptability of the model for different meteorological scenarios and for fields with various soil structure and field management, the fact that calibration and validation yield could have differences was already taken in consideration. But for this process to work, ideally the simulated yield should present:

- a range that is wider than the range of the observed data, in order to be familiar with the most extreme scenarios;
- a distribution of data that should cover with a certain degree of uniformity the whole spectrum, for the model not to be too heavily biased on certain yields.

While the first characteristic is verified, with the range of measured yield being contained by the simulated one, the distribution of the generated yield data seems to suggest that the training was mostly done on yields values close to the mean of the dataset (Figure 5a), which is a lot smaller than the case study measured one (0.90 tons/ha in calibration, versus 2.83 tons/ha in validation).

Another fact that makes the two sets of yield data substantially different is their constitution. The Daisy outcome is expressed as a value of Dry Matter substance, while the observed data is given in terms of fresh weight. To convert the measured quantities

in the analogue variable of the calibration dataset, a standard fresh weight to dry matter loss fraction of 14% was applied (Equation 10), but it could be source of error.

Most importantly, in situ yield data were not acquired with a rigorous procedure, with the yield measures being done also through a visual method at times as described in the ESYRCE methodology (M. Herrezuelo, 2023). This can be visually confirmed in Figure 11 with many points forming linear vertical patterns, caused by a large number of observations provided in steps of 0.5 tons/ha,

An a posteriori analysis supports the weak reliability in yields measured values. A random forest model was also trained on the observed in situ data, with the same set of twelve *operational* predictors (Table 6) used for model validation. The model shows a similar poor performance, with R^2 values of 0.402, RMSE equal to 0.755, and a Pearson coefficient of 0.636, especially if compared to the accuracy obtained with the same model architecture, trained on Daisy simulations data (Table 7).

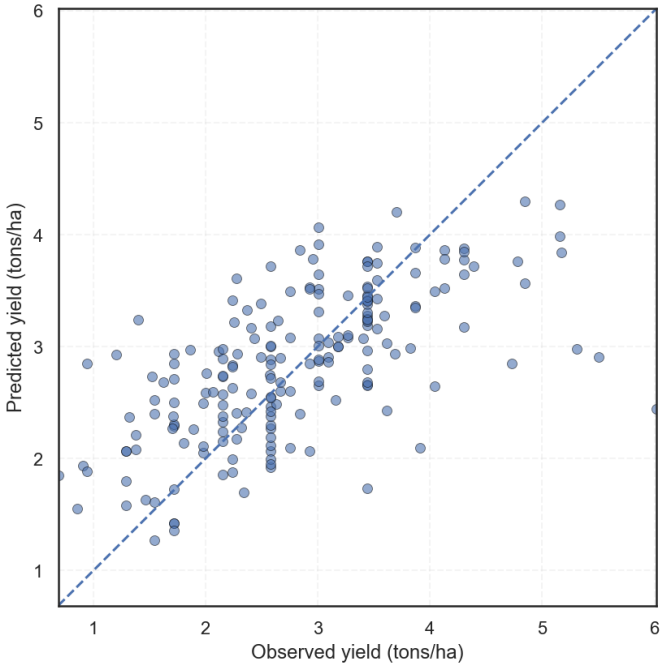


Figure 12: observed vs predicted yield scatter plot for model trained on observed in situ data

Regarding the simulations dataset used in the study, one crucial element could be the settings arranged for its generation (Algorithm 1). The harvest is set not to take place before the crop reaches quasi maturity (Development Stage = 1.9), with no option to make it happen at a certain date, regardless of the stage of the crop. This is most likely the reason behind the fact that of the 40 years theoretically included as meteorological years, only 23 are included in the output. In the other sets of years, the crop probably never reached DS = 1.9 before the end of the agronomic year. This could be a possible

factor causing the shape of the simulated yield curve to be concentrated on a small range of values.

5.1.2. Predictors

The previously mentioned setting of the Daisy simulations generator is influential in the harvest date of the simulations, and so in the values of the variables of the calibration dataset. As described for Figure 7a, *month_07_NP* is null for 94% of the simulations: this is due to the fact that the harvest was done in July only for 6% of the times.

A supposition is that, in Daisy's environment, wheat only reaches maturity if the crop has a quick growing dynamic, while not limited by Nitrogen deficit. The fact that the values of correlation of *days_0.X_var* indicators with the simulated yield are negative seem to support this thesis. On the other hand, in validation the correlation of the said variables (e.g. *days_0.9_WPlant*), is positive.

	Pearson (cal)	**Pearson (val)**
days_0.9_WPlant	-0.232	0.133

Figure 13: *days_0.9_WPlant* correlation with the output for calibration and for validation datasets

The sign discordance in correlation for the variable, if coupled with the fact that the variables have a central role in terms of importance in Random Forest model, could be one of the reasons for the model's low accuracy in the prediction.

Another cause for model's limited predictive ability attributable to the dataset was the exclusion in the *operational* of the some of the variables that explains yields variations best, such as the Net Photosynthesis related ones (Table 4) and plant Nitrogen content.

5.2. Model construction

Model constructing process must be held accountable as well, but in this case, it can be seen as a less important cause, since:

- both Random Forest and Symbolic Regression models, and machine learning models in general, are renowned for their accuracy and especially the ability to extract useful non-linear information;
- Random Forest model's performance in calibration (Table 7) shows quasi optimal values of the reported metrics, with an increase in the accuracy as the number of variables grows, and every Random Forest model outperforms linear regression corresponding one.

The machine learning models utilized, as shown in Figure 11, are more capable of understanding yield observations behaviour. On the other side, the Linear Regression approach has a higher value of Pearson correlation coefficient in validation compared to the Symbolic Regression. This is due to the fact that the first model is able to describe better the general trend of the observations but is strongly positive bias, having a value of mean bias of -1.762 tons/ha, a lot bigger than the one of Symbolic Regression, which stand at 0.330 tons/ha, with the mean bias being the difference between the mean of the predicted yield and the mean of the observed yield.

Even though, overall, the ability to predict yield of the Random Forest model outperforms the Symbolic Regression, the latter should not be discarded a priori, since it could be helpful for two reasons:

1. it can be used in parallel with the Random Forest model, to help in the predictor selection, since it shows which variables are more relevant for the describing the output;
2. it represents a simpler, more understandable model, that could have better applicability in a real case scenario.

6 Conclusions

This study investigated the potential of integrating crop simulation modelling and machine learning techniques for winter wheat yield mapping in central Spain. A large synthetic dataset was generated using the Daisy crop model, spanning multiple meteorological years and a wide range of soil and management conditions. This dataset was used to train Random Forest and Symbolic Regression models, which were then evaluated both within the synthetic domain and against real-world observations derived from Earth Observation data and field measurements.

Results obtained within the synthetic dataset show very high predictive performance, especially for Random Forest models, with R^2 values approaching 1 when using extended predictor sets. This confirms that machine learning algorithms are highly effective in extracting relationships embedded in physically based crop simulations and in capturing non-linear interactions among crop, soil, and meteorological variables.

However, when the calibrated models were applied to real-world observations, performance significantly decreased for all modelling approaches. Although Random Forest outperformed linear regression and Symbolic Regression in validation, all models exhibited lower explanatory power and relatively high prediction errors. This reduction in performance highlights the structural differences between the simulated environment and real agricultural conditions. However, the use of crop model simulations could still be a sound alternative in regions where in situ yield data is scarce enough for training a robust empirical model.

Several factors likely contributed to this discrepancy. Firstly, the distribution and magnitude of simulated yields differ substantially from observed yields, particularly in terms of mean value and variability. Secondly, some of the most influential predictors in the synthetic dataset (e.g. net photosynthesis and Nitrogen-related variables) were not available in the operational dataset, limiting the transferability of the calibrated models. Third, uncertainties in observed yield data and the necessary moisture correction introduce additional sources of error. Finally, assumptions embedded in the Daisy simulation setup, particularly regarding crop maturity and harvest timing, may have constrained the variability represented in the synthetic dataset.

Overall, the study demonstrates that the integration of crop modelling and machine learning represents a promising framework for yield mapping applications. The approach allows the exploitation of physically based knowledge while maintaining computational efficiency in large scale applications. Nevertheless, improving transferability to operational scenarios requires better alignment between simulated and observed yield distributions, refinement of crop model configurations, and the inclusion of remotely retrievable variables that more closely represent key physiological processes.

Future research should focus on refining the calibration of synthetic datasets, incorporating more realistic management variability, including additional Earth Observation products and/or methods related to photosynthesis and Nitrogen content (e.g. via relationships with chlorophyll concentration), and investigating methods to better align simulated data with real-world observations. Strengthening the consistency between modelling assumptions and field level observations is essential for enhancing the operational applicability of this integrated modelling framework.

Bibliography

Allen, R. G. . (2000). *Crop evapotranspiration : guidelines for computing crop water requirements*. F.A.O.

Arnold, J. G., Srinivasan, R., Muttiah, R. S., & Allen, P. M. (1999). CONTINENTAL SCALE SIMULATION OF THE HYDROLOGIC BALANCE [1]. *JAWRA Journal of the American Water Resources Association*, *35*(5), 1037–1051. https://doi.org/10.1111/j.1752-1688.1999.tb04192.x

Breiman, L. (2001). Random Forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (2017). *Classification And Regression Trees*. Routledge. https://doi.org/10.1201/9781315139470

Clevers, J. G. P. W. (1997). A simplified approach for yield prediction of sugar beet based on optical remote sensing data. *Remote Sensing of Environment*, *61*(2), 221–228. https://doi.org/10.1016/S0034-4257(97)00004-7

Copernicus.eu. (n.d.). *CORINE Land Cover*. Retrieved February 25, 2026, from https://land.copernicus.eu/en/products/corine-land-cover

Data Robot. (n.d.). *Eureqa Models*. Retrieved February 25, 2026, from https://docs.datarobot.com/en/docs/modeling/analyze-models/describe/eureqa.html

DE PURY, D. G. G., & FARQUHAR, G. D. (1997). Simple scaling of photosynthesis from leaves to canopies without the errors of big-leaf models. *Plant, Cell & Environment*, *20*(5), 537–557. https://doi.org/10.1111/j.1365-3040.1997.00094.x

Earth Data Hub, Destination EU. (n.d.). Retrieved February 24, 2026, from https://earthdatahub.destine.eu/

Gallego, J., Carfagna, E., & Baruth, B. (2010). Accuracy, Objectivity and Efficiency of Remote Sensing for Agricultural Statistics. In *Agricultural Survey Methods* (pp. 193–211). Wiley. https://doi.org/10.1002/9780470665480.ch12

Gardner, W. R. (1960). DYNAMIC ASPECTS OF WATER AVAILABILITY TO PLANTS. *Soil Science, 89*.

Guzinski, R., Nieto, H., Ramo Sánchez, R., Sánchez, J. M., Jomaa, I., Zitouna-Chebbi, R., Roupsard, O., & López-Urrea, R. (2023). Improving field-scale crop actual evapotranspiration monitoring with Sentinel-3, Sentinel-2, and Landsat data fusion. *International Journal of Applied Earth Observation and Geoinformation*, *125*, 103587. https://doi.org/10.1016/j.jag.2023.103587

Hansen, M. C., Potapov, P. V., Moore, R., Hancher, M., Turubanova, S. A., Tyukavina, A., Thau, D., Stehman, S. V., Goetz, S. J., Loveland, T. R., Kommareddy, A., Egorov, A., Chini, L.,

Justice, C. O., & Townshend, J. R. G. (2013). High-Resolution Global Maps of 21st-Century Forest Cover Change. *Science*, *342*(6160), 850–853. https://doi.org/10.1126/science.1244693

Hansen, S., Jensen, H. E., Nielsen, N. E., & Svendsen, H. (1990). *DAISY - Soil Plant Atmosphere System Model*.

Herman, J., & Usher, W. (2017). SALib: An open-source Python library for Sensitivity Analysis. *The Journal of Open Source Software*, *2*(9), 97. https://doi.org/10.21105/joss.00097

Hiederer, Roland. (2009). *Distribution of organic carbon in soil profile data*. OPOCE.

Holzworth, D. P., Huth, N. I., deVoil, P. G., Zurcher, E. J., Herrmann, N. I., McLean, G., Chenu, K., van Oosterom, E. J., Snow, V., Murphy, C., Moore, A. D., Brown, H., Whish, J. P. M., Verrall, S., Fainges, J., Bell, L. W., Peake, A. S., Poulton, P. L., Hochman, Z., … Keating, B. A. (2014). APSIM – Evolution towards a new generation of agricultural systems simulation. *Environmental Modelling & Software*, *62*, 327–350. https://doi.org/10.1016/j.envsoft.2014.07.009

Jaafar, H. H., Mourad, R. M., Kustas, W. P., & Anderson, M. C. (2022). A Global Implementation of Single- and Dual-Source Surface Energy Balance Models for Estimating Actual Evapotranspiration at 30-m Resolution Using Google Earth Engine. *Water Resources Research*, *58*(11). https://doi.org/10.1029/2022WR032800

Kustas, W. P., & Norman, J. M. (1999). Evaluation of soil and vegetation heat flux predictions using a simple two-source model with radiometric temperatures for partial canopy cover. *Agricultural and Forest Meteorology*, *94*(1), 13–29. https://doi.org/10.1016/S0168-1923(99)00005-2

Landsat Collection 2 Surface Temperature, United States Geological Survey. (n.d.). Retrieved February 24, 2026, from https://www.usgs.gov/landsat-missions/landsat-collection-2-surface-temperature

Lobell, D. B., Ortiz-Monasterio, J. I., Asner, G. P., Naylor, R. L., & Falcon, W. P. (2005). Combining Field Surveys, Remote Sensing, and Regression Trees to Understand Yield Variations in an Irrigated Wheat Landscape. *Agronomy Journal*, *97*(1), 241–249. https://doi.org/10.2134/agronj2005.0241a

Lobell, D. B., Thau, D., Seifert, C., Engle, E., & Little, B. (2015). A scalable satellite-based crop yield mapper. *Remote Sensing of Environment*, *164*, 324–333. https://doi.org/10.1016/j.rse.2015.04.021

MacDonald, R. B., & Hall, F. G. (1980). Global Crop Forecasting. *Science*, *208*(4445), 670–679. https://doi.org/10.1126/science.208.4445.670

Makke, N., & Chawla, S. (2024). Interpretable scientific discovery with symbolic regression: a review. *Artificial Intelligence Review*, *57*(1), 2. https://doi.org/10.1007/s10462-023-10622-0

MARKOVIĆ, M., FILIPOVIĆ, V., LEGOVIĆ, T., JOSIPOVIĆ, M., & TADIĆ, V. (2015). Evaluation of different soil water potential by field capacity threshold in combination with a triggered irrigation module. *Soil and Water Research*, *10*(3), 164–171. https://doi.org/10.17221/189/2014-SWR

MIGUEL ÁNGEL HERREZUELO BERMÚDEZ. (2023). *ESTIMACIÓN DEL RENDIMIENTO DE CEBADA Y TRIGO MEDIANTE IMÁGENES DE SENTINEL-2 EN LA PENÍNSULA IBÉRICA*.

Ministerio de Agricultura, P. y A. (n.d.). *Encuesta sobre Superficies y Rendimientos Cultivos (ESYRCE)*. Retrieved February 25, 2026, from https://www.mapa.gob.es/es/estadistica/temas/estadisticas-agrarias/agricultura/esyrce/

Mollerup, M., Abrahamsen, P., Petersen, C. T., & Hansen, S. (2014). Comparison of simulated water, nitrate, and bromide transport using a Hooghoudt-based and a dynamic drainage model. *Water Resources Research*, *50*(2), 1080–1094. https://doi.org/10.1002/2012WR013318

Norman, J. M., Kustas, W. P., & Humes, K. S. (1995). Source approach for estimating soil and vegetation energy fluxes in observations of directional radiometric surface temperature. *Agricultural and Forest Meteorology*, *77*(3–4), 263–293. https://doi.org/10.1016/0168-1923(95)02265-Y

Pedregosa & al. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research. *Journal of Machine Learning Research : JMLR*.

Richards, L. A. (1931). CAPILLARY CONDUCTION OF LIQUIDS THROUGH POROUS MEDIUMS. *Physics*, *1*(5), 318–333. https://doi.org/10.1063/1.1745010

Shanahan, J. F., Schepers, J. S., Francis, D. D., Varvel, G. E., Wilhelm, W. W., Tringe, J. M., Schlemmer, M. R., & Major, D. J. (2001). Use of Remote-Sensing Imagery to Estimate Corn Grain Yield. *Agronomy Journal*, *93*(3), 583–589. https://doi.org/10.2134/agronj2001.933583x

Sobol', I. M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, *7*(4), 86–112. https://doi.org/10.1016/0041-5553(67)90144-9

Strong, R. G., & Sbur, D. E. (1960). Influence of Grain Moisture and Storage Temperature on the Effectiveness of Malathion as a Grain Protectant1. *Journal of Economic Entomology*, *53*(3), 341–349. https://doi.org/10.1093/jee/53.3.341

The Cornell University AI for Science Institute (CUAISci). (n.d.). *Eureqa*. Retrieved February 25, 2026, from https://science.ai.cornell.edu/

Yagnik Pandya. (n.d.). *Ensemble Methods in Machine Learning*. Medium.Com. Retrieved February 24, 2026, from https://medium.com/analytics-vidhya/ensemble-methods-in-machine-learning-31084c3740be

A Appendix A: Daisy related scripts and templates

A.1. “daisy_simulations_template.dai”

```
;; Use standard parameterizations.
;; Search file files these places.
(path "C:/Users/aleka/OneDrive - Politecnico di Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-master_1/crop-model-simulations-master/resources/daisy"
"C:/Users/aleka/OneDrive - Politecnico di Milano/Documenti/Tesi/Daisy/daisy-7.1.0-win64/lib" &old)
(input file "tillage.dai")
(input file "crop.dai")
(input file "fertilizer.dai")
(input file "log.dai")
(input file "log_simulations.dai")

;; Weather data.
(weather default "<WEATHER>")

;; Suelo estándard en ávila
(defhorizon A FAO3
  "simulated"
  (clay <CLAY> [])
  (silt <SILT> [])
  (sand <SAND> [])
  (humus <HUMUS_A> [%])
  (C_per_N <C_PER_N> [g C/g N])
  (dry_bulk_density <BULK_DENSITY> [g/cm^3])
)

(defhorizon B A
  (humus <HUMUS_B> [%])
)
```

```
(defhorizon C A
  (humus 0.1 [%])
)

;; We build the column from the horizons.
(defcolumn "sim" default
  (Soil
    (MaxRootingDepth 100.0 [cm])
    (horizons
      (-30 [cm] A)
      (-100 [cm] B)
      (-400 [cm] C)
    )
  )
  (Groundwater deep)
  (Bioclimate default
    (pet PM)
    ;;(svat PMSW)
    ;;(ghf surface)
  )
)

;; Use it.
(column "sim")

(defcrop "Maize_SP" "Silage Maize"
  (Devel original
    ;(EmrTSum 200)
    ;(DSRate1  0.024) (DSRate2  0.021)
  )
)

(defaction maize activity
  (wait_mm_dd 04 15)
  (wait (or trafficable (mm_dd 5 15)))
  (fertilize
    (AmmoniumNitrate (weight <N_PRECISION> [kg N/ha]))
  )
```

```
  (seed_bed_preparation)
  (sow "Maize_SP")
  (wait (crop_ds_after "Maize_SP" 0.25))
  (fertilize
    (AmmoniumNitrate (weight <N_PRECISION> [kg N/ha]))
  )
  (wait (or (crop_ds_after "Maize_SP" 2.0)(mm_dd 11 01)))
  (harvest "Maize_SP"
    (stub 8 [cm])
  )
  (wait_mm_dd 11 15)
  (fertilize
    ("cattle_manure" (weight 40 [T w.w./ha]))(to -5 [cm])
  )
)


(defaction spring_wheat activity
  (wait_mm_dd 04 15)
  (wait (or trafficable (mm_dd 5 15)))
  (fertilize
    (NPK02 (weight <N_PRECISION> [kg N/ha]))
  )
 (seed_bed_preparation)
 (sow "Spring Wheat")
 (wait (crop_ds_after "Spring Wheat" 0.2))
 (fertilize
   (NPK02 (weight <N_PRECISION> [kg N/ha]))
 )
 (wait (or (crop_ds_after "Spring Wheat" 2.0)(mm_dd 9 30)))
 (harvest "Spring Wheat"
   (stub 8 [cm])
 )
)

(defaction irrigate activity
  (wait (not (soil_water_pressure_above
              (height -10 [cm])
              (potential -33 [kPa]))))
```

```
  (irrigate_surface 5 [mm/h])
  (wait_days <W_PRECISION>)
)

(defaction winter_wheat activity
  (wait_mm_dd 09 01)
  (wait (or trafficable (mm_dd 10 15)))
  (fertilize
    (NPK02 (weight <N_PRECISION> [kg N/ha]))
  )
 (seed_bed_preparation)
 (sow "Winter Wheat")
 (wait (crop_ds_after "Winter Wheat" 0.2))
 (fertilize
   (NPK02 (weight <N_PRECISION> [kg N/ha]))
 )
 (activity (repeat irrigate))
 (wait (crop_ds_after "Winter Wheat" 1.9))
 (harvest "Winter Wheat"
   (stub 8 [cm])
 )
)

;; Simulation start and stop dates.
(time 1981 1 1)
(stop 2020 12 31)
(manager activity
  (repeat <CROP>)
)

;; Create these log files.
(activate_output (after 1981 1 1))
(output
  (harvest
    (where "<HARVEST_OUTPUT>")
  )
  ("Crop Photosynthesis"
    (when monthly)
    (where "<CO2_OUTPUT>"))
```

```
  ("Crop Biophysical"
    (where "<CROP_OUTPUT>"))
  ("Fluxes"
    (when monthly)
    (where "<H2O_OUTPUT>"))
)
```

A.2. "get_daisy_simulations.py"

```
from pathlib import Path
import datetime as dt
import calendar
import matplotlib.pyplot as plt
from crop_model_helpers.daisy import daisy
import crop_model_helpers.handlers as hd
import logging

n_proc = -1
n_simulations = 1000
overwrite = False
random_seed = 1000
delete_outputs = False
plot_results = True
site_name = "iberia-central"
workdir = Path().absolute() / "resources"  / "daisy"
outdir = workdir / "simulations"
crop = "winter_wheat"
config_file_template = workdir / \
                       "daisy_simulations_template.dai"

daisy.DAISY_EXE = Path().home() / "OneDrive - Politecnico di Milano" /
"Documenti" / "Tesi" / "Daisy" / "daisy-7.1.0-win64" / "bin" / "daisy"

output_file = outdir / f"daisy_yield_simulations_{site_name}_{crop}.csv"
weather_file = config_file_template.parent / f"{site_name}.dwf"

bounds =(                       (0.04, 0.45), # clay
                                (0.26, 0.94), # sand
                                (0.926, 1.527), # bulk density
                                (0.5, 3.0), # hummus horizon A
```

```
                                (0.05, 0.9), # mummus horizon B
                                (6.8, 17.2), # C:N
                                (0, 50), # Nitrogen management
                                (1, 29) # Alternate wetting days
                                )

# Ensure that the names of the variables match the
# <replacement string> in the daisy configuration template file
SENSITIVITY_PROBLEM = {'groups': ("texture",
                                  "texture",
                                  "density",
                                  "humus",
                                  "humus",
                                  "C/N",
                                  "management",
                                  "management"),
                       'names': ("CLAY",
                                 "SAND",
                                 "BULK_DENSITY",
                                 "HUMUS_A",
                                 "HUMUS_B",
                                 "C_PER_N",
                                 "N_PRECISION",
                                 "W_PRECISION")
                       }

def make_plots(results):
    valid = results["sorg_DM"] >= 0

    # Plot monthly cummulative metrics
    for var in ["NetPhotosynthesis", "T"]:
        fig, ax = plt.subplots(4, 3, sharex="col", sharey=True)
        ax = ax.reshape(-1)

        for i, m in enumerate(range(1, 13)):
            ax[i].scatter(results.loc[valid, f"month_{m:02}_{var}"],
                          results.loc[valid, "sorg_DM"],
                          color="b", marker=".", alpha=0.1)
            ax[i].set_title(calendar.month_abbr[m])
```

```
        fig.suptitle(var)
        plt.subplots_adjust(hspace=0.1, wspace=0)

    # Plot Daily-based metrics
    for var in ["WPlant", "NPlant", "LAI"]:
        fig, ax = plt.subplots(2, 4, sharey=True)
        ax = ax.reshape(-1)
        for i, m in enumerate(["mean", "std", "max"]):
            ax[i].scatter(results.loc[valid, f"{m}_{var}"],
                          results.loc[valid, "sorg_DM"],
                          color="b", marker=".", alpha=0.1)
            ax[i].set_xlabel(m)

        for j, m in enumerate([0.1, 0.2, 0.5, 0.8, 0.9]):
            ax[j + i + 1].scatter(results.loc[valid, f"days_{m}_{var}"],
                                  results.loc[valid, "sorg_DM"],
                                  color="b", marker=".", alpha=0.1)
            ax[j + i + 1].set_xlabel(F"Days above {m:.0%} peak")

        plt.subplots_adjust(wspace=0, hspace=0.2)
        fig.suptitle(var)

    plt.show()


if __name__ == "__main__":
    if not outdir.is_dir():
        outdir.mkdir(parents=True)

    logger = logging.getLogger()
    logfilename = f"{Path(__file__).stem}_{dt.datetime.today().date()}"
    logfile_path = Path() / "logs" / f"{logfilename}.log"
    errorfile_path = Path() / "logs" / f"{logfilename}.err"
    logger = hd.prepare_log(logger,
                            logfile_path=logfile_path,
                            errorfile_path=errorfile_path)

    problem = SENSITIVITY_PROBLEM
```

```
    problem['num_vars'] = len(problem["names"])
    problem["bounds"] = bounds

    logger.info(f"Running Monte Carlo Daisy simulations "
                f"based on {config_file_template}\n"
                f"using {weather_file} weather\n"
                f"and with Saltelli configuration: {problem}")

    results = daisy.run_daily_simulations(problem,
                                          outdir,
                                          crop,
                                          config_file_template,
                                          n_simulations,
                                          weather_file,
                                          output_file,
                                          n_proc=n_proc,
                                          overwrite=overwrite,
                                          delete_outputs=delete_outputs,
                                          random_seed=random_seed)

    if plot_results:
        make_plots(results)
```

A.3. Single simulation output: "sim.txt"

```
;; Use standard parameterizations.
;; Search file files these places.
(path "C:/Users/aleka/OneDrive - Politecnico di Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-master_1/crop-model-simulations-master/resources/daisy"
"C:/Users/aleka/OneDrive - Politecnico di Milano/Documenti/Tesi/Daisy/daisy-7.1.0-win64/lib" &old)
(input file "tillage.dai")
(input file "crop.dai")
(input file "fertilizer.dai")
(input file "log.dai")
(input file "log_simulations.dai")

;; Weather data.
```

```
(weather default "C:/Users/aleka/OneDrive - Politecnico di Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-master_1/crop-model-simulations-master/resources/daisy/central-iberia.dwf")

;; Suelo estándard en ávila
(defhorizon A FAO3
  "simulated"
  (clay 0.3313470454327762 [])
  (silt 0.20124962062869628 [])
  (sand 0.46740333393852757 [])
  (humus 0.5803736946545541 [%])
  (C_per_N 16.047723867744207 [g C/g N])
  (dry_bulk_density 1.380922798141837 [g/cm^3])
)

(defhorizon B A
  (humus 0.1842955269402288 [%])
)

(defhorizon C A
  (humus 0.1 [%])
)

;; We build the column from the horizons.
(defcolumn "sim" default
  (Soil
    (MaxRootingDepth 100.0 [cm])
    (horizons
      (-30 [cm] A)
      (-100 [cm] B)
      (-400 [cm] C)
    )
  )
  (Groundwater deep)
  (Bioclimate default
    (pet PM)
    ;;(svat PMSW)
    ;;(ghf surface)
  )
```

```
)

;; Use it.
(column "sim")

(defcrop "Maize_SP" "Silage Maize"
  (Devel original
    ;(EmrTSum 200)
    ;(DSRate1  0.024) (DSRate2  0.021)
  )
)

(defaction maize activity
  (wait_mm_dd 04 15)
  (wait (or trafficable (mm_dd 5 15)))
  (fertilize
    (AmmoniumNitrate (weight 28.57689382508397 [kg N/ha]))
  )
  (seed_bed_preparation)
  (sow "Maize_SP")
  (wait (crop_ds_after "Maize_SP" 0.25))
  (fertilize
    (AmmoniumNitrate (weight 28.57689382508397 [kg N/ha]))
  )
  (wait (or (crop_ds_after "Maize_SP" 2.0)(mm_dd 11 01)))
  (harvest "Maize_SP"
    (stub 8 [cm])
  )
  (wait_mm_dd 11 15)
  (fertilize
    ("cattle_manure" (weight 40 [T w.w./ha]))(to -5 [cm])
  )
)


(defaction spring_wheat activity
  (wait_mm_dd 04 15)
  (wait (or trafficable (mm_dd 5 15)))
  (fertilize
```

```
    (NPK02 (weight 28.57689382508397 [kg N/ha]))
  )
 (seed_bed_preparation)
 (sow "Spring Wheat")
 (wait (crop_ds_after "Spring Wheat" 0.2))
 (fertilize
   (NPK02 (weight 28.57689382508397 [kg N/ha]))
 )
 (wait (or (crop_ds_after "Spring Wheat" 2.0)(mm_dd 9 30)))
 (harvest "Spring Wheat"
   (stub 8 [cm])
 )
)

(defaction irrigate activity
  (wait (not (soil_water_pressure_above
              (height -10 [cm])
              (potential -33 [kPa]))))
  (irrigate_surface 5 [mm/h])
  (wait_days 133)
)

(defaction winter_wheat activity
  (wait_mm_dd 09 01)
  (wait (or trafficable (mm_dd 10 15)))
  (fertilize
    (NPK02 (weight 28.57689382508397 [kg N/ha]))
  )
 (seed_bed_preparation)
 (sow "Winter Wheat")
 (wait (crop_ds_after "Winter Wheat" 0.2))
 (fertilize
   (NPK02 (weight 28.57689382508397 [kg N/ha]))
 )
 (activity (repeat irrigate))
 (wait (crop_ds_after "Winter Wheat" 1.9))
 (harvest "Winter Wheat"
   (stub 8 [cm])
 )
```

```
)

;; Simulation start and stop dates.
(time 1981 1 1)
(stop 2020 12 31)
(manager activity
  (repeat winter_wheat)
)

;; Create these log files.
(activate_output (after 1981 1 1))
(output
  (harvest
    (where "C:/Users/aleka/OneDrive - Politecnico di
Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-
master_1/crop-model-simulations-
master/resources/daisy/simulations/pool/0/harvest.dlf")
  )
  ("Crop Photosynthesis"
    (when monthly)
    (where "C:/Users/aleka/OneDrive - Politecnico di
Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-
master_1/crop-model-simulations-
master/resources/daisy/simulations/pool/0/co2.dlf"))
  ("Crop Biophysical"
    (where "C:/Users/aleka/OneDrive - Politecnico di
Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-
master_1/crop-model-simulations-
master/resources/daisy/simulations/pool/0/crop.dlf"))
  ("Fluxes"
    (when monthly)
    (where "C:/Users/aleka/OneDrive - Politecnico di
Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-
master_1/crop-model-simulations-
master/resources/daisy/simulations/pool/0/h2o.dlf"))
)
```

A.1. Single simulation output: “co2.txt”

```
dlf-0.0 -- Crop Photosynthesis (defined in 'log.dai').

VERSION: 7.1.0
```

```
LOGFILE:       C:/Users/aleka/OneDrive       -       Politecnico       di
Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-
master_1/crop-model-simulations-
master/resources/daisy/simulations/pool/0/co2.dlf
RUN: Tue Dec  9 11:07:32 2025


COLUMN: *
CROP: *
INTERVAL: box none


LOG: A log table for a specific crop.


SIMFILE:       C:\Users\aleka\OneDrive       -       Politecnico       di
Milano\Documenti\Tesi\PROGETTO\01_data_gen\crop-model-simulations-
master_1\crop-model-simulations-
master\resources\daisy\simulations\pool\0\sim.dai


--------------------
year  month mday  hour  NetPhotosynthesis RootRespiration   CanopyAss
      PotCanopyAss      Bioinc_CO2  OM_CO2
                        g CO2/m^2   g CO2/m^2   g CH2O/m^2  g CH2O/m^2  g
C/m^2 g CO2-C/cm^2
1998  2     1     0     00.00 00.00 00.00 00.00 0     0.00249562
1998  3     1     0     00.00 00.00 00.00 00.00 0     0.00127394
1998  4     1     0     00.00 00.00 00.00 00.00 0     0.000991007
1998  5     1     0     00.00 00.00 00.00 00.00 0     0.00104086
1998  6     1     0     00.00 00.00 00.00 00.00 0     0.00097484
1998  7     1     0     00.00 00.00 00.00 00.00 0     0.000785344
1998  8     1     0     00.00 00.00 00.00 00.00 0     0.000620696
1998  9     1     0     00.00 00.00 00.00 00.00 0     0.000517585
1998  10    1     0     50.842      6.62901     43.8279     43.8279     0
      0.000479016
1998  11    1     0     864.264     114.323     830.497     830.497     0
      0.000515849
1998  12    1     0     568.564     171.347     737.469     1519.82     0
      0.000695812
1999  1     1     0     -126.377    139.998     126.042     1623.81     0
      0.000728776
1999  2     1     0     -148.498    116.134     82.0691     1578.82     0
      0.000735531
1999  3     1     0     -152.013    86.6609     47.0311     1154.47     0
      0.000689224
1999  4     1     0     -195.402    82.1947     35.2232     653.914     0
      0.000896445
```

```
1999  5     1     0     -3.90086    75.4896     146.058     538.34      0
      0.000976595
```

A.2. Single simulation output: “co2.txt”

```
dlf-0.0 -- Fluxes (defined in 'log_simulations.dai').


VERSION: 7.1.0

LOGFILE:        C:/Users/aleka/OneDrive        -        Politecnico        di
Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-
master_1/crop-model-simulations-
master/resources/daisy/simulations/pool/0/h2o.dlf

RUN: Tue Dec  9 11:07:32 2025


COLUMN: *

INTERVAL: box none


LOG: Summary of water fluxes.


SIMFILE:        C:\Users\aleka\OneDrive        -        Politecnico        di
Milano\Documenti\Tesi\PROGETTO\01_data_gen\crop-model-simulations-
master_1\crop-model-simulations-
master\resources\daisy\simulations\pool\0\sim.dai


--------------------
year  month mday  hour  ET_ref      ET_0  ET    T_0   T
                        mm    mm    mm    mm    mm
1998  2     1     0     221.199     133.081     61.1804     43.1405     0
1998  3     1     0     234.761     141.197     17.3037     74.3363     0
1998  4     1     0     283.432     170.503     10.7097     95.8763     0
1998  5     1     0     220.437     133.408     42.347      54.6366     0
1998  6     1     0     199.9 121.959     87.8375     20.473      0
1998  7     1     0     167.725     104.275     101.926     1.40941     0
1998  8     1     0     147.731     94.5307     93.4609     0.641866    0
1998  9     1     0     118.449     80.0175     80.0175     8.32667e-18 0
1998  10    1     0     121.086     106.376     102.605     2.34334
      2.34334
1998  11    1     0     139.888     153.494     152.466     65.976
      65.976
1998  12    1     0     143.943     154.114     107.75      144.277
      98.2249
```

```
1999  1     1     0     137.945     144.493     17.6675     136.378
      13.4185
1999  2     1     0     149.618     153.24      10.5949     143.053
      7.47037
1999  3     1     0     144.457     148.215     10.9108     133.415
      5.27227
1999  4     1     0     179.257     184.222     11.3099     158.526
      3.70777
1999  5     1     0     176.942     182.636     42.9712     127.971
      6.40841
```

A.3. Single simulation output: “crop.txt”

```
dlf-0.0 -- Fluxes (defined in 'log_simulations.dai').


VERSION: 7.1.0

LOGFILE:        C:/Users/aleka/OneDrive        -        Politecnico        di
Milano/Documenti/Tesi/PROGETTO/01_data_gen/crop-model-simulations-
master_1/crop-model-simulations-
master/resources/daisy/simulations/pool/0/h2o.dlf

RUN: Tue Dec  9 11:07:32 2025


COLUMN: *

INTERVAL: box none


LOG: Summary of water fluxes.


SIMFILE:        C:\Users\aleka\OneDrive        -        Politecnico        di
Milano\Documenti\Tesi\PROGETTO\01_data_gen\crop-model-simulations-
master_1\crop-model-simulations-
master\resources\daisy\simulations\pool\0\sim.dai


--------------------
year  month mday  hour  ET_ref      ET_0  ET    T_0   T
                        mm    mm    mm    mm    mm
1998  2     1     0     221.199     133.081     61.1804     43.1405     0
1998  3     1     0     234.761     141.197     17.3037     74.3363     0
1998  4     1     0     283.432     170.503     10.7097     95.8763     0
1998  5     1     0     220.437     133.408     42.347      54.6366     0
1998  6     1     0     199.9 121.959     87.8375     20.473      0
1998  7     1     0     167.725     104.275     101.926     1.40941     0
1998  8     1     0     147.731     94.5307     93.4609     0.641866    0
```

```
1998  9     1     0     118.449     80.0175     80.0175     8.32667e-18 0
1998  10    1     0     121.086     106.376     102.605     2.34334
      2.34334
1998  11    1     0     139.888     153.494     152.466     65.976
      65.976
1998  12    1     0     143.943     154.114     107.75      144.277
      98.2249
1999  1     1     0     137.945     144.493     17.6675     136.378
      13.4185
1999  2     1     0     149.618     153.24      10.5949     143.053
      7.47037
1999  3     1     0     144.457     148.215     10.9108     133.415
      5.27227
1999  4     1     0     179.257     184.222     11.3099     158.526
      3.70777
1999  5     1     0     176.942     182.636     42.9712     127.971
      6.40841
```

B Appendix B: Model training and validation

B.1. "expl_var.py"

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jan 15 17:58:19 2026

@author: aleka
"""

# -*- coding: utf-8 -*-
"""
Created on Thu Nov 13 13:44:19 2025

@author: aleka
"""
"""
--> in case I want to use a similar folder directory structure as Héctor
site_name = "Italia"
crop = "winter_wheat"
"""




#
%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
_°_
# 0) DATA IMPORT AND DIRECTORY PATHWAY SELECTION

# %% Library import
```

```
import pandas as pd
import numpy as np
# import seaborn as sns
# --> per grafici avanzati
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split
from scipy.stats import pearsonr, spearmanr
import joblib
import time
from pathlib import Path
from scipy.stats import skew, kurtosis

from scipy.spatial.distance import squareform
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error

import re

import seaborn as sns



# %% Directory

workdir = Path(r"C:\Users\aleka\OneDrive - Politecnico di
Milano\Documenti\Tesi\PROGETTO\02_model_train\expl_var")
indir = workdir / "daisy_out"



#%% Data import

start = time.time()
input_df = pd.read_csv(indir / "daisy_yield_simulations_iberia-
central_winter_wheat.csv", sep = ";")
```

```
print(f"Data loaded in {time.time() - start:.2f} s")
del start

# %%  Rows shuffle
# --> to make sure there is statistical coherence in the train/test subsets
input_df = input_df.sample(frac=1, random_state=42).reset_index(drop=True)

# %% "NetPhotosynthesis" --> "NP"

input_df.columns = [
    col.replace("NetPhotosynthesis", "NP")
    for col in input_df.columns
]



# %% Create X_df and Y (keep input_df intact)

X_df = input_df.copy()
Y = X_df["sorg_DM"]





#%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
°_°_
# 1) PRELIMINARY STEPS

# %% Analysis and Elimination of extreme "sorg_DM" values
'''
mask_ok = Y <= 10

n_gt_10 = (~mask_ok).sum()
perc_gt_10 = (~mask_ok).mean() * 100

descr_extr = Y[~mask_ok].describe()
descr_extr["variance"] = descr_extr["std"]**2
descr_extr["|CV|"] = abs(descr_extr["std"]/descr_extr["mean"])
```

```
print("Total number of output values higher than 10: ", n_gt_10)
print("Equal to: ", f"{perc_gt_10:.2f}%")

X_df = X_df.loc[mask_ok].reset_index(drop=True)
Y = Y.loc[mask_ok].reset_index(drop=True)

del n_gt_10, perc_gt_10, mask_ok
'''
mask_ok = X_df["month"] <= 6
n_gt_6 = (~mask_ok).sum()
perc_gt_6 = (~mask_ok).mean() * 100
print("Total number of simulations with month > 6:", n_gt_6)
print("Equal to:", f"{perc_gt_6:.2f}%")



# %% Check column types

not_double_cols = X_df.select_dtypes(exclude=["float", "int"]).columns
print(not_double_cols.tolist())


'''
cols_to_convert = [c for c in not_double_cols if c != "crop"]

for c in cols_to_convert:
    X_df[c] = pd.to_numeric(X_df[c], errors="coerce")

X_df = X_df.drop(columns=["crop"]) # removing "crop" column
del not_double_cols, cols_to_convert, c
'''

# %% Check missing values

missing = X_df.isna().sum()
missing_cols = missing[missing > 0]

if missing_cols.empty:
    print("No missing values were found in the dataset")
```

```
else:
    print("Columns with missing values:")
    print(missing_cols)
# "month_10_T" have 77 missing values; I will remove all the lines where these
# values where present

'''
mask_ok = X_df["month_10_T"].notna()

X_df = X_df.loc[mask_ok].reset_index(drop=True)
Y = Y.loc[mask_ok].reset_index(drop=True)

del missing, missing_cols, mask_ok
'''

# %% Group variables

# Components
components_WTotal = ["sorg_DM", "stem_DM", "leaf_DM", "dead_DM"]
components_NTotal = ["sorg_N", "stem_N", "leaf_N", "dead_N"]

# Output Y (not to remove)
protect_cols = ["sorg_DM"]

# WTotal
available_W = [c for c in components_WTotal if c in X_df.columns]

if len(available_W) >= 2:
    X_df["WTotal"] = X_df[available_W].sum(axis=1)
    print(f"WTotal created using: {available_W}")

    # remove only components NOT in protect list
    cols_to_drop = [c for c in available_W if c not in protect_cols]
    X_df = X_df.drop(columns=cols_to_drop)
    print(f"Removed W components (excluding protected ones): {cols_to_drop}")
else:
    print("WTotal NOT created (too few components available)")
```

```
# NTotal
available_N = [c for c in components_NTotal if c in X_df.columns]

if len(available_N) >= 2:
    X_df["NTotal"] = X_df[available_N].sum(axis=1)
    print(f"NTotal created using: {available_N}")

    cols_to_drop = [c for c in available_N if c not in protect_cols]
    X_df = X_df.drop(columns=cols_to_drop)
    print(f"Removed N components: {cols_to_drop}")
else:
    print("NTotal NOT created (too few components available)")

del available_N, available_W, components_NTotal, components_WTotal,
protect_cols, cols_to_drop


# %% Remove of "Harvest time" variables
harv_var = [
    "stem_DM", "dead_DM", "leaf_DM", "stem_N", "dead_N", "leaf_N", "sorg_N",
    "PAI", "LAI", "WLeaf", "WPlant", "NLeaf", "NPlant", "WTotal", "NTotal"
]

avail_harv_var = [c for c in harv_var if c in X_df.columns]
X_df = X_df.drop(columns=avail_harv_var)

print("These Harvest Variables are removed:", avail_harv_var)
del harv_var, avail_harv_var


# %% Remove all variables containing "Leaf"
leaf_cols = [c for c in X_df.columns if "Leaf" in c]

X_df = X_df.drop(columns=leaf_cols)

print("Columns removed (contain 'Leaf'):", leaf_cols)

del leaf_cols
```

```
# %% Creation of predictors (X) and output (Y)
Y = Y.to_numpy()

# all the columns besides inputs for Daisy and yield output are potential predictors
cols = list(X_df.columns)
X_names = cols[cols.index("mean_PAI"):]
X = X_df[X_names].to_numpy()
#       pd.DataFrame([X_names]).to_csv("X_names_row.csv",       index=False, header=False)
# --> printing of the X_names names

del cols

# %% Remove constant variables
# I remove variables which have 0 variability --> they can never explain the yield
range_vals = np.ptp(X, axis = 0)
const_idx = range_vals == 0
X = X[:, ~const_idx]
X_names = np.array(X_names)[~const_idx].tolist()

print(f"Removed   {np.sum(const_idx)}   constant   variables.   Remaining: {len(X_names)}")
del range_vals, const_idx


# %% Remove PAI variables (perfectly correlated with LAI)

mask_no_pai = np.array(["PAI" not in name for name in X_names])

X = X[:, mask_no_pai]
X_names = np.array(X_names)[mask_no_pai].tolist()

print(f"Removed PAI variables. Remaining predictors: {len(X_names)}")
```

```
# %% Create OPERATIONAL predictors (without NPlant, NP and non-validatable
vars)

remove_terms = ["NPlant", "NP"]

remove_exact = [

    # LAI timing not available
    "doy_max_LAI",
    "days_0.2_LAI",
    "days_0.8_LAI",

    # WPlant timing not available
    "doy_max_WPlant",
    "days_0.2_WPlant",
    "days_0.8_WPlant",

    # ET_ref not available from satellite
    "month_10_ET_ref",
    "month_11_ET_ref",
    "month_12_ET_ref",
    "month_01_ET_ref",
    "month_02_ET_ref",
    "month_03_ET_ref",
    "month_04_ET_ref",
    "month_05_ET_ref",
    "month_06_ET_ref",
    "month_07_ET_ref",
    "month_08_ET_ref",
    "month_09_ET_ref",
]

keep_mask = np.array([
    (not any(term in name for term in remove_terms)) and (name not in
remove_exact)
    for name in X_names
])

X_names_op = np.array(X_names)[keep_mask].tolist()
```

```
X_op = X[:, keep_mask]
input_df_op = X_df[X_names_op].copy()

print(f"Operational predictors: {len(X_names_op)} / {len(X_names)}")

# check if there's any leakage between the variables and the output
suspects = [c for c in X_names_op if ("sorg" in c.lower()) or ("yield" in c.lower())]
print("Suspect cols:", suspects)









#%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
° °
 _ _
# 2) DESCRIPTIVE ANALYSIS

# %% Basic statistics
# includes: mean, std, min, max, 25/50/75 percentiles; adding variance and CV
desc_basics = pd.DataFrame(X, columns=X_names).describe().T
desc_basics["variance"] = desc_basics["std"]**2
desc_basics["|CV|"] = abs(desc_basics["std"]/desc_basics["mean"])

desc_basics_Y = pd.Series(Y, name="sorg_DM").describe()
desc_basics_Y["variance"] = desc_basics_Y["std"]**2
desc_basics_Y["|CV|"] = abs(desc_basics_Y["std"]/desc_basics_Y["mean"])

# %% Boxplot & histogram of output

# BASIC
plt.figure(figsize=(6,4))
plt.hist(X_df["sorg_DM"], bins=30)
plt.title("sorg_DM")
```

```
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()

plt.figure(figsize=(6,4))
plt.hist(X_df["sorg_DM"], bins=30)
plt.yscale("log")
plt.title("sorg_DM")
plt.xlabel("Value")
plt.ylabel("Frequency")
plt.show()

plt.figure(figsize=(6,4))
plt.boxplot(X_df["sorg_DM"], vert=True, patch_artist=True)
plt.title("Boxplot: sorg_DM")
plt.ylabel("Value")
plt.show()


#%% GOOD LOOKING
sns.set(style="whitegrid")

plt.figure(figsize=(6,4))
sns.histplot(X_df["sorg_DM"], bins=30, kde=True)

plt.xlabel("Yield (tons/ha)")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()



plt.figure(figsize=(6,4))
sns.histplot(X_df["sorg_DM"], bins=30, kde=False)

plt.yscale("log")

plt.title("Distribution of sorg_DM (log scale)")
```

```
plt.xlabel("Yield (t/ha)")
plt.ylabel("Frequency (log)")

plt.tight_layout()
plt.show()



sns.set(style="whitegrid")
plt.figure(figsize=(4,6))
sns.boxplot(y=X_df["sorg_DM"])

plt.ylabel("Yield (tons/ha)")

plt.tight_layout()
plt.show()



plt.figure(figsize=(4,6))

sns.boxplot(y=X_df["sorg_DM"], color="lightblue")
sns.stripplot(y=X_df["sorg_DM"], color="black", alpha=0.3)

plt.title("sorg_DM distribution with observations")
plt.ylabel("Yield (t/ha)")

plt.tight_layout()
plt.show()



sample = X_df["sorg_DM"].sample(n=10000, random_state=42)
plt.figure(figsize=(6,4))
sns.violinplot(y=X_df["sorg_DM"], color="lightblue")
sns.stripplot(y=sample, color="black", size=2, alpha=0.3)
plt.title("sorg_DM")
plt.show()
```

```
plt.figure(figsize=(6,4))
sns.boxplot(y=X_df["sorg_DM"], color="lightgray")
sns.stripplot(y=X_df["sorg_DM"], color="black", size=1, alpha=0.1)
plt.title("sorg_DM")
plt.show()

skew_Y = skew(Y)
# positive values --> longer tail on the right;
# negative values --> longer tail on the left
# absolute value > 1 --> important skewness
kurt_Y = kurtosis(Y)



mask_ok = X_df["month_07_NP"] == 0
m_0 = (mask_ok).sum()
perc_m_0 = (mask_ok).mean() * 100
print("Total number of simulations with month_07_NP == 0:", m_0)
print("Equal to:", f"{perc_m_0:.2f}%")



# %% Skewness & Kurtosis
skew_vals = skew(X, axis=0, nan_policy = "omit")
# positive values --> longer tail on the right;
# negative values --> longer tail on the left
# absolute value > 1 --> important skewness
kurt_vals = kurtosis(X, axis=0, nan_policy = "omit")
# < 3 --> flat distribution
# > 3 --> sharp

desc_shape = pd.DataFrame({
    "Variable": X_names,
    "Skewness": skew_vals,
    "Kurtosis": kurt_vals
})
```

```
desc_shape = desc_shape.reindex(desc_shape["Skewness"].abs()
                                .sort_values(ascending=False).index)

# most interesting variables --> very skewed;
high_skew = desc_shape[desc_shape["Skewness"].abs() > 1]
high_skew = high_skew.reset_index(drop=True)

# I want to see the boxplot and the histogramm of these
vars_to_plot = high_skew["Variable"].iloc[:10].tolist()

for var in vars_to_plot:
    plt.figure(figsize=(6,4))
    plt.hist(X_df[var], bins=30)
    plt.yscale("log")
    plt.title(f"Skewed: {var}")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    plt.show()
    plt.figure(figsize=(6,4))
    plt.boxplot(X_df[var], vert=True, patch_artist=True)
    plt.title(f"Boxplot: {var}")
    plt.ylabel("Value")
    plt.show()

del skew_vals, kurt_vals, vars_to_plot

# %% Outliers analysis
# outlier = datas having being bigger or smaller than the mean +/- three
times the std

outlier_table = []
for i, var in enumerate(X_names):
    data = X[:, i]
    mu = data.mean()
    sigma = data.std()
    outliers = data[(data < mu - 3*sigma) | (data > mu + 3*sigma)]
    outlier_table.append({
        "Variable": var,
```

```
        "Min": data.min(),
        "Max": data.max(),
        "Mean": mu,
        "Std": sigma,
        "CV": sigma/mu,
        "n_outliers": len(outliers)
    })

outlier_df = pd.DataFrame(outlier_table)
outlier_df             =             outlier_df.sort_values(by="n_outliers",
ascending=False).reset_index(drop=True)

vars_to_plot = outlier_df["Variable"].iloc[:10].tolist()

for var in vars_to_plot:
    plt.figure(figsize=(6,4))
    plt.hist(X_df[var], bins=30)
    plt.yscale("log")
    plt.title(f"Outlier: {var}")
    plt.xlabel("Value")
    plt.ylabel("Frequency (log scale)")
    plt.show()
    plt.figure(figsize=(6,4))
    plt.boxplot(X_df[var], vert=True, patch_artist=True)
    plt.title(f"Boxplot: {var}")
    plt.ylabel("Value")
    plt.show()

del data, mu, sigma, outliers, outlier_table, vars_to_plot
```

```
#%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
°_°_
```

```
# 3) CORRELATION
# I compute the correlation of the variables with the output and the cross-correlation
# between variables, with the Pearson coefficient (stronger: looking for linear corr)
# and with Spearman (weaker: looking for ranking correlation)

X_train, X_test, Y_train, Y_test = train_test_split(
    X,
    Y,
    test_size=0.2,
    shuffle=False
)



# %% PEARSON correlation

rho_P = np.array([pearsonr(X_train[:,i], Y_train)[0] for i in range(X_train.shape[1])])
pval_P = np.array([pearsonr(X_train[:, i], Y_train)[1] for i in range(X_train.shape[1])])

T_rho_P = pd.DataFrame({"Variable": X_names, "Pearson": rho_P, "pValue": pval_P})
#T_rho_P = T_rho_P.reindex(T_rho_P["Pearson"].abs().sort_values(ascending=False).index)
T_rho_P = T_rho_P.reset_index(drop = True)
T_rho_P.index = T_rho_P.index + 1

# T_rho_P.to_csv(workdir / "T_rho_P.csv", index=True)
# --> I can print the results for looking at them better

# statistically robust
mask_P = (pval_P < 0.01)
rob_P = T_rho_P[mask_P].reset_index(drop = True)



# %% SPEARMAN correlation
rho_S = np.array([spearmanr(X_train[:,i], Y_train)[0] for i in range(X_train.shape[1])])
```

```
pval_S = np.array([spearmanr(X_train[:,i], Y_train)[1] for i in range(X_train.shape[1])])


T_rho_S = pd.DataFrame({"Variable": X_names, "Spearman": rho_S, "p_value": pval_S})
#T_rho_S = T_rho_S.reindex(T_rho_S["Spearman"].abs().sort_values(ascending = False).index)
T_rho_S = T_rho_S.reset_index(drop = True)
T_rho_S.index = T_rho_S.index + 1


# statistically robust
mask_S = (pval_S < 0.01)
rob_S = T_rho_S[mask_S].reset_index(drop = True)



# %% top variables


# Merge of the correlation in a single dataframe
T_corr = pd.DataFrame({
    "Variable": X_names,
    "Pearson": rho_P,
    "p_Pearson": pval_P,
    "Spearman": rho_S,
    "p_Spearman": pval_S
})


T_corr["Score"] = np.maximum(np.abs(T_corr["Pearson"]), np.abs(T_corr["Spearman"]))
T_corr_sorted = T_corr.sort_values(by="Score", ascending=False).reset_index(drop=True)


top_k = 50
T_top50 = T_corr_sorted.iloc[:top_k].copy()


best_50_vars = T_top50["Variable"].tolist()
best_50_idx = [X_names.index(v) for v in best_50_vars]
X_top50_train = X_train[:, best_50_idx]
X_top50_test  = X_test[:, best_50_idx]


# %% Alphabetical order
```

```
# I put the top50 elements in alphabetical order so that it will be easier
to
# read the cross correlation matrix
sorted_vars = sorted(best_50_vars) # alphabetical
sorted_idx = [best_50_vars.index(v) for v in sorted_vars]
X_top50_train = X_top50_train[:, sorted_idx]
X_top50_test  = X_top50_test[:, sorted_idx]
T_top50_sorted                                                          =
T_top50.set_index("Variable").loc[sorted_vars].reset_index()

# Std, for linear regression
scaler = StandardScaler()
X_top50_train_std = scaler.fit_transform(X_top50_train)
X_top50_test_std = scaler.transform(X_top50_test)

# %% cross correlation

corr_P = np.corrcoef(X_top50_train, rowvar = False)

plt.figure(figsize=(8, 6))
plt.imshow(np.abs(corr_P), cmap="viridis", interpolation="nearest")
plt.colorbar(label="|ρ|")
plt.title("Pearson (|ρ|)")

plt.xticks(
    ticks=np.arange(len(sorted_vars)),
    labels=sorted_vars,
    rotation=90,
    fontsize=5
)
plt.yticks(
    ticks=np.arange(len(sorted_vars)),
    labels=sorted_vars,
    fontsize=5
)

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

```
# top50 where: |Pearson| > |Spearman|
top50_pearson_dom = T_top50[np.abs(T_top50["Pearson"]) > np.abs(T_top50["Spearman"])]
vars_pearson_dom = top50_pearson_dom["Variable"].tolist()
print("Variables where |P|>|S|: ", vars_pearson_dom)

'''
%% cross correlation Spearman
'''





#%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
# 4) DENDROGRAMS and VARIABLES SELECTION
# I can do a dendrogram to look at how the variables split in different groups.
# Then for examples I can take a variable out of each group.

# Dendrogram of relation of variables with output --> usefull only to see how
# variables are correlated with the output


#%% PEARSON dendrogram (TOP VAR)
D = 1 - np.abs(corr_P)
D_condensed = squareform(D, checks=False)
Z_P_top = linkage(D_condensed, method='average')

plt.figure(figsize=(12, 5))
dendrogram(Z_P_top, labels=sorted_vars, leaf_rotation = 90)
yticks = np.arange(0, plt.gca().get_ylim()[1], 0.01)
plt.yticks(yticks)
plt.grid(axis='y', linestyle='--', color='gray', alpha=0.3)
plt.show()
```

```
# %% 2 vars selection
clusters_2vars = fcluster(Z_P_top, t=2, criterion='maxclust')


selected_2vars = []
for c in np.unique(clusters_2vars):
    # variables indexes in the cluster
    idx_cluster = np.where(clusters_2vars == c)[0]
    # selecting variable with highest correlation
    best_idx                                                                =
idx_cluster[np.argmax(np.abs(T_top50_sorted["Pearson"].values[idx_cluster])
)]
    selected_2vars.append(sorted_vars[best_idx])


print("Selected variables (best correlated with Y per each cluster):",
selected_2vars)



# %% 4 vars selection
clusters_4vars = fcluster(Z_P_top, t=4, criterion='maxclust')


selected_4vars = []
for c in np.unique(clusters_4vars):
    idx_cluster = np.where(clusters_4vars == c)[0]
    best_idx                                                                =
idx_cluster[np.argmax(np.abs(T_top50_sorted["Pearson"].values[idx_cluster])
)]
    selected_4vars.append(sorted_vars[best_idx])


print("Selected variables (best correlated with Y per each cluster):",
selected_4vars)



# %% 8 vars selection
clusters_8vars = fcluster(Z_P_top, t=8, criterion='maxclust')


selected_8vars = []
for c in np.unique(clusters_8vars):
    idx_cluster = np.where(clusters_8vars == c)[0]
    best_idx                                                                =
idx_cluster[np.argmax(np.abs(T_top50_sorted["Pearson"].values[idx_cluster])
)]
    selected_8vars.append(sorted_vars[best_idx])
```

```
print("Selected variables (best correlated with Y per each cluster):",
selected_8vars)



# %% 12 vars selection
clusters_12vars = fcluster(Z_P_top, t=12, criterion='maxclust')


selected_12vars = []
for c in np.unique(clusters_12vars):
    idx_cluster = np.where(clusters_12vars == c)[0]
    best_idx                                                                =
idx_cluster[np.argmax(np.abs(T_top50_sorted["Pearson"].values[idx_cluster])
)]
    selected_12vars.append(sorted_vars[best_idx])


print("Selected variables (best correlated with Y per each cluster):",
selected_12vars)



# %% Scatter Plot delle 12 variabili con l'output
for var in selected_12vars:
    plt.figure(figsize=(5,4))
    plt.scatter(X_df[var], X_df["sorg_DM"], s=5, alpha=0.5)
    plt.xlabel(var)
    plt.ylabel("sorg_DM")
    plt.title(f"{var} vs sorg_DM")
    plt.tight_layout()
    plt.show()



'''
%% PEARSON dendrogram (ALL VAR)
'''



'''
%% SPEARMAN dendrogram (TOP VAR)
'''
```

```
#%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
# 5) RANDOM FOREST
# I now want to build a model that predicts the output starting from the
# selected variables; I will confront the performance of the random forest
# against a abseline model of a simple linear regression


#%% # Linear regression function
def run_linear_regression(X_train, X_test, y_train, y_test, feature_names):

    lr = LinearRegression()
    lr.fit(X_train, y_train)

    # predictions
    y_pred = lr.predict(X_test)

    # metrics
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    r_pearson, _ = pearsonr(y_test, y_pred)
    r2_pearson = r_pearson**2
    bias = np.mean(y_test) - np.mean(y_pred)

    # coefficients
    coeffs = pd.DataFrame({
        "Variable": feature_names,
        "Coefficient": lr.coef_
    }).sort_values(by="Coefficient", key=np.abs, ascending=False)

    return r2, rmse, r_pearson, r2_pearson, bias, coeffs, lr
```

```
#%% # RF function
# We first define a function that: 1) creates the training and testing dataset,
#


def run_random_forest(X_train, X_test, y_train, y_test, feature_names, random_state=98):

    rf = RandomForestRegressor(
        n_estimators=500,
        max_depth=None,
        min_samples_leaf=2,
        random_state=random_state,
        n_jobs=-1
    )

    rf.fit(X_train, y_train)

    # predictions
    y_pred = rf.predict(X_test)

    # metrics
    r2 = r2_score(y_test, y_pred)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))

    r_pearson, _ = pearsonr(y_test, y_pred)
    r2_pearson = r_pearson**2
    bias = np.mean(y_test) - np.mean(y_pred)

    # importance
    importance = pd.DataFrame({
        "Variable": feature_names,
        "Importance": rf.feature_importances_
    }).sort_values(by="Importance", ascending=False)

    return r2, rmse, r_pearson, r2_pearson, bias, importance, rf
```

```
#%% ===== 2 VARS =====

idx_2 = [sorted_vars.index(v) for v in selected_2vars]

X_2_train = X_top50_train[:, idx_2]
X_2_test  = X_top50_test[:, idx_2]

X_2_train_std = X_top50_train_std[:, idx_2]
X_2_test_std  = X_top50_test_std[:, idx_2]

# Linear regression
start = time.time()
r2_lr2,    rmse_lr2,    r_lr2,    r2p_lr2,    bias_lr2,    coef_lr2,    lr2=
run_linear_regression(
    X_2_train_std,          X_2_test_std,          Y_train,          Y_test,
feature_names=selected_2vars
)

print(f"Time {time.time() - start:.2f} s")
print(f"LR (2 vars) --> R2 = {r2_lr2:.3f}, RMSE = {rmse_lr2:.3f}, Pearson =
{r_lr2:.3f}, Pearson2 = {r2p_lr2:.3f}, Bias = {bias_lr2:.3f}")
print(coef_lr2)
del start

# Random Forest
start = time.time()
r2_2, rmse_2, r_rf2, r2p_rf2, bias_rf2, imp_2, rf2 = run_random_forest(
    X_2_train, X_2_test, Y_train, Y_test, feature_names=selected_2vars
)

print(f"Time {time.time() - start:.2f} s")
print(f"RF (2 vars) --> R2 = {r2_2:.3f}, RMSE = {rmse_2:.3f}, Pearson =
{r_rf2:.3f}, Pearson2 = {r2p_rf2:.3f}, Bias = {bias_rf2:.3f}")
print(imp_2)
del start



#%% ===== 4 VARS =====

idx_4 = [sorted_vars.index(v) for v in selected_4vars]
```

```
X_4_train = X_top50_train[:, idx_4]
X_4_test  = X_top50_test[:, idx_4]

X_4_train_std = X_top50_train_std[:, idx_4]
X_4_test_std  = X_top50_test_std[:, idx_4]

# Linear regression
start = time.time()
r2_lr4,   rmse_lr4,   r_lr4,   r2p_lr4,   bias_lr4,   coef_lr4,   lr4   =
run_linear_regression(
    X_4_train_std,          X_4_test_std,          Y_train,          Y_test,
feature_names=selected_4vars
)

print(f"Time {time.time() - start:.2f} s")
print(f"LR (4 vars) --> R2 = {r2_lr4:.3f}, RMSE = {rmse_lr4:.3f}, Pearson =
{r_lr4:.3f}, Pearson2 = {r2p_lr4:.3f}, Bias = {bias_lr4:.3f}")
print(coef_lr4)
del start

# Random Forest
start = time.time()
r2_4, rmse_4, r_rf4, r2p_rf4, bias_rf4, imp_4, rf4 = run_random_forest(
    X_4_train, X_4_test, Y_train, Y_test, feature_names=selected_4vars
)

print(f"Time {time.time() - start:.2f} s")
print(f"RF  (4 vars)  --> R2  = {r2_4:.3f}, RMSE = {rmse_4:.3f}, Pearson =
{r_rf4:.3f}, Pearson2 = {r2p_rf4:.3f}, Bias = {bias_rf4:.3f}")
print(imp_4)
del start



#%% ===== 8 VARS =====

idx_8 = [sorted_vars.index(v) for v in selected_8vars]

X_8_train = X_top50_train[:, idx_8]
X_8_test  = X_top50_test[:, idx_8]
```

```
X_8_train_std = X_top50_train_std[:, idx_8]
X_8_test_std  = X_top50_test_std[:, idx_8]


# Linear regression
start = time.time()
r2_lr8,   rmse_lr8,   r_lr8,   r2p_lr8,   bias_lr8,   coef_lr8,   lr8   =
run_linear_regression(
    X_8_train_std,          X_8_test_std,          Y_train,          Y_test,
feature_names=selected_8vars
)


print(f"Time {time.time() - start:.2f} s")
print(f"LR (8 vars) --> R2 = {r2_lr8:.3f}, RMSE = {rmse_lr8:.3f}, Pearson =
{r_lr8:.3f}, Pearson2 = {r2p_lr8:.3f}, Bias = {bias_lr8:.3f}")
print(coef_lr8)
del start


# Random Forest
start = time.time()
r2_8, rmse_8, r_rf8, r2p_rf8, bias_rf8, imp_8, rf8 = run_random_forest(
    X_8_train, X_8_test, Y_train, Y_test, feature_names=selected_8vars
)


print(f"Time {time.time() - start:.2f} s")
print(f"RF (8 vars) --> R2 = {r2_8:.3f}, RMSE = {rmse_8:.3f}, Pearson =
{r_rf8:.3f}, Pearson2 = {r2p_rf8:.3f}, Bias = {bias_rf8:.3f}")
print(imp_8)
del start



#%% ===== 12 VARS =====


idx_12 = [sorted_vars.index(v) for v in selected_12vars]


X_12_train = X_top50_train[:, idx_12]
X_12_test  = X_top50_test[:, idx_12]


X_12_train_std = X_top50_train_std[:, idx_12]
X_12_test_std  = X_top50_test_std[:, idx_12]
```

```
# Linear regression
start = time.time()
r2_lr12, rmse_lr12, r_lr12, r2p_lr12, bias_lr12, coef_lr12, lr12 = run_linear_regression(
    X_12_train_std, X_12_test_std, Y_train, Y_test, feature_names=selected_12vars
)

print(f"Time {time.time() - start:.2f} s")
print(f"LR (12 vars) --> R2 = {r2_lr12:.3f}, RMSE = {rmse_lr12:.3f}, Pearson = {r_lr12:.3f}, Pearson2 = {r2p_lr12:.3f}, Bias = {bias_lr12:.3f}")
print(coef_lr12)
del start

# Random Forest
start = time.time()
r2_12, rmse_12, r_rf12, r2p_rf12, bias_rf12, imp_12, rf12= run_random_forest(
    X_12_train, X_12_test, Y_train, Y_test, feature_names=selected_12vars
)

print(f"Time {time.time() - start:.2f} s")
print(f"RF (12 vars) --> R2 = {r2_12:.3f}, RMSE = {rmse_12:.3f}, Pearson = {r_rf12:.3f}, Pearson2 = {r2p_rf12:.3f}, Bias = {bias_rf12:.3f}")
print(imp_12)
del start



'''
#%% ===== top vars =====
'''
```

```
#%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
# OPERATIVE
# From correlation onwards, using ONLY operational predictors:
# X_op, X_names_op, input_df_op
# (Target is still Y -> simulated output)

#%% Train/test split (operational predictors, SAME Y as before)

X_train_op, X_test_op, Y_train, Y_test = train_test_split(
    X_op,
    Y,
    test_size=0.2,
    shuffle=False
)


#%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
# 3) CORRELATION (OPERATIVE)

# %% PEARSON correlation (operational)
rho_P_op = np.array([pearsonr(X_train_op[:, i], Y_train)[0] for i in range(X_train_op.shape[1])])
pval_P_op = np.array([pearsonr(X_train_op[:, i], Y_train)[1] for i in range(X_train_op.shape[1])])

T_rho_P_op = pd.DataFrame({"Variable": X_names_op, "Pearson": rho_P_op, "pValue": pval_P_op})
T_rho_P_op = T_rho_P_op.reset_index(drop=True)
T_rho_P_op.index = T_rho_P_op.index + 1

mask_P_op = (pval_P_op < 0.01)
rob_P_op = T_rho_P_op[mask_P_op].reset_index(drop=True)


# %% SPEARMAN correlation (operational)
```

```
rho_S_op = np.array([spearmanr(X_train_op[:, i], Y_train)[0] for i in range(X_train_op.shape[1])])
pval_S_op = np.array([spearmanr(X_train_op[:, i], Y_train)[1] for i in range(X_train_op.shape[1])])


T_rho_S_op = pd.DataFrame({"Variable": X_names_op, "Spearman": rho_S_op, "p_value": pval_S_op})
T_rho_S_op = T_rho_S_op.reset_index(drop=True)
T_rho_S_op.index = T_rho_S_op.index + 1


mask_S_op = (pval_S_op < 0.01)
rob_S_op = T_rho_S_op[mask_S_op].reset_index(drop=True)



# %% Top variables (operational)
T_corr_op = pd.DataFrame({
    "Variable": X_names_op,
    "Pearson": rho_P_op,
    "p_Pearson": pval_P_op,
    "Spearman": rho_S_op,
    "p_Spearman": pval_S_op
})


T_corr_op["Score"] = np.maximum(np.abs(T_corr_op["Pearson"]), np.abs(T_corr_op["Spearman"]))
T_corr_sorted_op = T_corr_op.sort_values(by="Score", ascending=False).reset_index(drop=True)


top_k_op = 50
T_top50_op = T_corr_sorted_op.iloc[:top_k_op].copy()


best_50_vars_op = T_top50_op["Variable"].tolist()
best_50_idx_op = [X_names_op.index(v) for v in best_50_vars_op]


X_top50_train_op = X_train_op[:, best_50_idx_op]
X_top50_test_op  = X_test_op[:, best_50_idx_op]



# %% Alphabetical order (operational)
sorted_vars_op = sorted(best_50_vars_op)
```

```
sorted_idx_op = [best_50_vars_op.index(v) for v in sorted_vars_op]

X_top50_train_op = X_top50_train_op[:, sorted_idx_op]
X_top50_test_op  = X_top50_test_op[:, sorted_idx_op]

T_top50_sorted_op                                                    =
T_top50_op.set_index("Variable").loc[sorted_vars_op].reset_index()

# Std for linear regression (operational)
scaler_op = StandardScaler()
X_top50_train_std_op = scaler_op.fit_transform(X_top50_train_op)
X_top50_test_std_op  = scaler_op.transform(X_top50_test_op)



# %% Cross correlation (operational)
corr_P_op = np.corrcoef(X_top50_train_op, rowvar=False)

plt.figure(figsize=(8, 6))
plt.imshow(np.abs(corr_P_op), cmap="viridis", interpolation="nearest")
plt.colorbar(label="|ρ|")
plt.title("Pearson (|ρ|) - OPERATIVE")

plt.xticks(
    ticks=np.arange(len(sorted_vars_op)),
    labels=sorted_vars_op,
    rotation=90,
    fontsize=5
)
plt.yticks(
    ticks=np.arange(len(sorted_vars_op)),
    labels=sorted_vars_op,
    fontsize=5
)

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

```
top50_pearson_dom_op = T_top50_op[np.abs(T_top50_op["Pearson"]) > np.abs(T_top50_op["Spearman"])]

vars_pearson_dom_op = top50_pearson_dom_op["Variable"].tolist()

print("Variables where |P|>|S| (OPERATIVE): ", vars_pearson_dom_op)



#%%°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_°_
# 4) DENDROGRAMS and VARIABLES SELECTION (OPERATIVE)

#%% PEARSON dendrogram (TOP VAR - OPERATIVE)
D_op = 1 - np.abs(corr_P_op)
D_condensed_op = squareform(D_op, checks=False)
Z_P_top_op = linkage(D_condensed_op, method='average')

plt.figure(figsize=(12, 5))
dendrogram(Z_P_top_op, labels=sorted_vars_op, leaf_rotation=90)
yticks = np.arange(0, plt.gca().get_ylim()[1], 0.01)
plt.yticks(yticks)
plt.grid(axis='y', linestyle='--', color='gray', alpha=0.3)
plt.show()



# %% 12 vars selection (OPERATIVE)
clusters_12vars_op = fcluster(Z_P_top_op, t=12, criterion='maxclust')

selected_12vars_op = []
for c in np.unique(clusters_12vars_op):
    idx_cluster = np.where(clusters_12vars_op == c)[0]
    best_idx = idx_cluster[np.argmax(np.abs(T_top50_sorted_op["Pearson"].values[idx_cluster]))]
    selected_12vars_op.append(sorted_vars_op[best_idx])

print("Selected variables (best correlated with Y per each cluster) - OPERATIVE:", selected_12vars_op)



#%% ===== 12 VARS (OPERATIVE) =====
```

```
idx_12_op = [sorted_vars_op.index(v) for v in selected_12vars_op]


X_12_train_op = X_top50_train_op[:, idx_12_op]
X_12_test_op  = X_top50_test_op[:, idx_12_op]


X_12_train_std_op = X_top50_train_std_op[:, idx_12_op]
X_12_test_std_op  = X_top50_test_std_op[:, idx_12_op]


# Linear regression (OPERATIVE)
start = time.time()
r2_lr12_op,    rmse_lr12_op,    r_lr12_op,    r2p_lr12_op,    bias_lr12_op,
coef_lr12_op, model_lr12_op = run_linear_regression(
    X_12_train_std_op,       X_12_test_std_op,       Y_train,       Y_test,
feature_names=selected_12vars_op
)



print(f"Time {time.time() - start:.2f} s")
print(f"LR  (12  vars)  -  OPERATIVE  -->  R2  =  {r2_lr12_op:.3f},  RMSE  =
{rmse_lr12_op:.3f}, Pearson = {r_lr12_op:.3f}, Pearson2 = {r2p_lr12_op:.3f},
Bias = {bias_lr12_op:.3f}")
print(coef_lr12_op)
del start


# Random Forest (OPERATIVE)
start = time.time()
r2_12_op,  rmse_12_op,  r_rf12_op,  r2p_rf12_op,  bias_rf12_op,  imp_12_op,
model_rf12_op = run_random_forest(
    X_12_train_op,          X_12_test_op,          Y_train,          Y_test,
feature_names=selected_12vars_op
)


print(f"Time {time.time() - start:.2f} s")
print(f"RF  (12  vars)  -  OPERATIVE  -->  R2  =  {r2_12_op:.3f},  RMSE  =
{rmse_12_op:.3f}, Pearson = {r_rf12_op:.3f}, Pearson2 = {r2p_rf12_op:.3f},
Bias = {bias_rf12_op:.3f}")
print(imp_12_op)
del start


joblib.dump(model_lr12_op, "lr_model_12vars_op.pkl")
joblib.dump(model_rf12_op, "rf_model_12vars_op.pkl")
```

```
joblib.dump(scaler_op, "scaler_top50_op.pkl")
joblib.dump(sorted_vars_op, "vars_top50_sorted_op.pkl")
joblib.dump(selected_12vars_op, "vars_12_op.pkl")
```

```
# %% 7) VALIDATION
```

```
# %% Validation data import
val_dir = Path(r"C:\Users\aleka\OneDrive - Politecnico di Milano\Documenti\Tesi\PROGETTO\03_model_val\yield_observations_spain\Alessandro")

start = time.time()
val_monthly_raw = pd.read_csv(val_dir / "wheat_bio_ET_monthly.csv", sep=";")
val_annual_raw  = pd.read_csv(val_dir / "wheat_bio_annual.csv", sep=";")
print(f"Validation data loaded in {time.time() - start:.2f} s")
del start

keys = ["ID", "YEAR"]
```

```
#%% Utility: duplicate report

def dup_report(df, name, keys):
    dup_rows = df.duplicated(keys, keep=False).sum()
    counts = df.groupby(keys).size()
    dup_keys = (counts > 1).sum()
    dist = counts[counts > 1].value_counts().sort_index()

    print(f"\n--- {name} ---")
```

```
    print("Rows total:", len(df))
    print("Duplicate-key rows:", int(dup_rows))
    print("N. duplicate keys:", int(dup_keys))
    if int(dup_keys) > 0:
        print("Duplicate multiplicity (n_rows_per_key -> n_keys):")
        print(dist.to_string())

#%% BEFORE collapsing
dup_report(val_monthly_raw, "MONTHLY (before collapse)", keys)
dup_report(val_annual_raw,  "ANNUAL  (before collapse)", keys)



#%% 1) Collapse duplicates by (ID, YEAR) taking mean of non-NA numeric values

def collapse_by_mean(df, keys):
    df = df.copy()

    # numeric columns to average (excluding keys)
    num_cols = df.select_dtypes(include="number").columns.difference(keys)

    # non-numeric columns: keep first non-null (crop/tile labels, etc.)
    other_cols = [c for c in df.columns if c not in num_cols and c not in
keys]

    # numeric: mean ignoring NaN (if all NaN -> NaN)
    g_num = df.groupby(keys, as_index=False)[list(num_cols)].mean()

    def first_non_null(s):
        s2 = s.dropna()
        return s2.iloc[0] if len(s2) else np.nan

    if other_cols:
        g_other = (df.groupby(keys, as_index=False)[other_cols]
                     .agg(first_non_null))
        out = g_num.merge(g_other, on=keys, how="left")
    else:
        out = g_num

    # keys first
```

```
    return out[[*keys] + [c for c in out.columns if c not in keys]]

val_monthly = collapse_by_mean(val_monthly_raw, keys)
val_annual  = collapse_by_mean(val_annual_raw, keys)

print("\nAfter collapsing:")
print("monthly rows:", len(val_monthly))
print("annual rows :", len(val_annual))

#%% AFTER collapsing
dup_report(val_monthly, "MONTHLY (after collapse)", keys)
dup_report(val_annual,  "ANNUAL  (after collapse)", keys)

#%% hard check: no duplicate keys remain
assert not val_monthly.duplicated(keys).any()
assert not val_annual.duplicated(keys).any()



#%% 2) Check YIELD consistency between the two datasets (same ID & YEAR)
#    (done AFTER collapsing, BEFORE merge)

def yield_consistency_check(df_m, df_a, keys, tol=0.0, max_examples=10):
    # yield columns in each df (case-insensitive)
    m_ycols = [c for c in df_m.columns if "YIELD" in c.upper()]
    a_ycols = [c for c in df_a.columns if "YIELD" in c.upper()]

    print("\n=== YIELD consistency check (monthly vs annual) ===")
    print("Monthly yield-like columns:", m_ycols)
    print("Annual  yield-like columns:", a_ycols)

    if (len(m_ycols) == 0) or (len(a_ycols) == 0):
        print("No yield columns found in one of the datasets -> skipping
check.")
        return

    # compare only common yield column names (same base name)
    common_y = sorted(set(m_ycols).intersection(a_ycols))
    if len(common_y) == 0:
```

```
        print("No yield columns with the SAME name in both datasets -> skipping check.")
        return

    # merge only keys + common yields
    m_sub = df_m[keys + common_y].copy()
    a_sub = df_a[keys + common_y].copy()
    chk = m_sub.merge(a_sub, on=keys, how="inner", suffixes=("_m", "_a"))

    print("Common (ID,YEAR) rows used for yield-check:", len(chk))

    for y in common_y:
        c_m = f"{y}_m"
        c_a = f"{y}_a"

        s1 = chk[c_m]
        s2 = chk[c_a]

        both = s1.notna() & s2.notna()
        n_both = int(both.sum())

        if n_both == 0:
            print(f"- {y}: no rows where both are non-NA -> skipped")
            continue

        diff = (s1[both] - s2[both]).abs()
        n_bad = int((diff > tol).sum())

        print(f"- {y}: compared {n_both} rows, mismatches = {n_bad} (tol={tol})")

        if n_bad > 0:
            bad_rows = chk.loc[both, keys + [c_m, c_a]].copy()
            bad_rows["abs_diff"] = diff.values
            bad_rows = bad_rows.sort_values("abs_diff", ascending=False)

            print("  Examples (largest diffs):")
            print(bad_rows.head(max_examples).to_string(index=False))
```

```
yield_consistency_check(val_monthly, val_annual, keys, tol=0.0, max_examples=10)



#%% 3) Merge (avoid column collisions)


common_cols = set(val_monthly.columns).intersection(val_annual.columns) - set(keys)
val_annual_ren = val_annual.rename(columns={c: f"{c}_annual" for c in common_cols})


merged = val_monthly.merge(val_annual_ren, on=keys, how="inner")
print("\nmerged rows:", len(merged))


merged_names = merged.columns.tolist()
print("\nN. merged columns:", len(merged_names))



# check on FW0p90M-BIOMASS values
fw = merged["FW0p90M-BIOMASS"]


print("min,max:", fw.min(), fw.max())







#%% 4) Keep only operational validation variables + save output


merged_df = merged.copy()


# --- 4.1) Save output (YIELD) ---
# prefer "YIELD" if present, otherwise fall back to "YIELD_annual"
yield_candidates = [c for c in ["YIELD", "YIELD_annual"] if c in merged_df.columns]
if len(yield_candidates) == 0:
    raise KeyError("No YIELD column found in merged_df (expected 'YIELD' or 'YIELD_annual').")


ycol = yield_candidates[0]
```

```
Y_val = merged_df[ycol].copy()
print(f"Using '{ycol}' as validation output (Y_val).")



# --- 4.2) Columns to drop (non-operational) ---
# IDs/metadata/output columns (we keep output_val separately)
drop_exact = [
    "ID", "YEAR",
    "CROP", "CROP_annual",
    "TILE", "TILE_bio",
    "YIELD", "YIELD_annual"
]

# Drop families (patterns)
drop_prefixes = [
    "ETac_", "water_def_", "annual_",
]

# Drop monthly families not used in cal
# (CAB, CCC, NDVI, CWSI, TWSI, ratios)
drop_month_suffix = ["_CAB", "_CCC", "_NDVI", "_CWSI", "_TWSI", "_R_T_ET",
"_R_T0_ET0"]

# --- 4.3) Handle FW columns: keep only the ones we can match with cal ---

fw_keep = {"FW0p10M-LAI", "FW0p50M-LAI", "FW0p90M-LAI",  "FW0p10M-BIOMASS",
"FW0p50M-BIOMASS", "FW0p90M-BIOMASS"}

fw_cols = [c for c in merged_df.columns if c.startswith("FW")]

# drop all FW* except the 3 LAI ones and the 3 of biomass
drop_fw = [c for c in fw_cols if c not in fw_keep]

# --- 4.4) Build final drop list safely ---
drop_cols = set()

# exact
drop_cols.update([c for c in drop_exact if c in merged_df.columns])
```

```
# prefixes
for p in drop_prefixes:
    drop_cols.update([c for c in merged_df.columns if c.startswith(p)])

# --- keep NAN* only for BIOMASS and LAI ---
nan_cols = [c for c in merged_df.columns
            if c.startswith(("NANMAX-", "NANMEAN-", "NANSTD-"))]

nan_keep = [c for c in nan_cols if c.endswith(("BIOMASS", "LAI"))]
nan_drop = set(nan_cols) - set(nan_keep)

drop_cols.update(nan_drop)

# monthly suffix families (e.g. "10_CAB", "1_CWSI", etc.)
# Works because your columns are like "10_CAB", "11_CWSI", ...
for suff in drop_month_suffix:
    drop_cols.update([c for c in merged_df.columns if c.endswith(suff)])

# FW drops
drop_cols.update(drop_fw)

# also drop FW for non-LAI even if naming differs (extra safety)
# (already covered by drop_fw)

# --- 4.4.bis) Drop months 8 and 9 for variables missing in calibration ---
# (in calibration you don't have month_08/month_09 for LAI, WPlant~BIOMASS,
T and T_0)
drop_months_8_9_suffix = ["_LAI", "_BIOMASS", "_T-gf"]
drop_months_8_9 = []

for m in ["8_", "9_"]:  # validation columns are like "8_LAI", "9_T-gf", ...
    for suff in drop_months_8_9_suffix:
        drop_months_8_9 += [c for c in merged_df.columns if c.startswith(m)
and c.endswith(suff)]

drop_cols.update(drop_months_8_9)



# --- 4.5) Apply drop ---
```

```
merged_op = merged_df.drop(columns=sorted(drop_cols), errors="ignore")


print(f"Columns before: {merged_df.shape[1]}")
print(f"Columns after : {merged_op.shape[1]}")
print(f"Dropped       : {len(drop_cols)}")









#%% 5) Unit conversions (UdM)
# Convert ONLY true BIOMASS in kg/ha -> t/ha:
# - monthly biomass columns like "10_BIOMASS", "1_BIOMASS", ...
# - NAN* biomass columns like "NANMEAN-BIOMASS", "NANMAX-BIOMASS", "NANSTD-BIOMASS"
# DO NOT convert FW* columns (they are timing/threshold metrics, not biomass amounts)
'''
print("\nFW0p90M-BIOMASS BEFORE conversion:")
print(merged_op["FW0p90M-BIOMASS"].describe())
'''


monthly_biomass_cols = [c for c in merged_op.columns if re.match(r"^\d{1,2}_BIOMASS$", c)]
nan_biomass_cols = [c for c in merged_op.columns if c.startswith(("NANMAX-", "NANMEAN-", "NANSTD-")) and c.endswith("BIOMASS")]


biomass_cols_to_convert = monthly_biomass_cols + nan_biomass_cols


print("BIOMASS cols to convert (kg/ha -> t/ha):", biomass_cols_to_convert)


if biomass_cols_to_convert:
    merged_op.loc[:, biomass_cols_to_convert] = merged_op[biomass_cols_to_convert] / 1000.0
    print(f"Converted {len(biomass_cols_to_convert)} BIOMASS columns from kg/ha to t/ha.")
else:
    print("No BIOMASS columns found to convert.")


'''
```

```
print("\nFW0p90M-BIOMASS AFTER conversion:")
print(merged_op["FW0p90M-BIOMASS"].describe())
'''
#%% 6) Rename columns to match CAL naming

# creation of a copy not to disrupt the original df
merged_op_try = merged_op.copy()

rename_map = {}

# --- (A) Monthly variables: "<m>_<VAR>" or "<m>_<VAR>-gf" ->
"month_<mm>_<CALVAR>"
month_var_map = {
    "BIOMASS": "WPlant",
    "LAI": "LAI",
    "T_0": "T_0",
    "T": "T",
    "ET_0": "ET_0",
    "ET": "ET",
}

for col in merged_op_try.columns:
    # accept both "10_ET-gf" and "10_ET"
    if "-gf" in col:
        base = col.replace("-gf", "")
    else:
        base = col

    # match pattern like "10_BIOMASS", "1_ET_0", etc.
    if "_" in base:
        left, right = base.split("_", 1)
        if left.isdigit() and 1 <= int(left) <= 12:
            mm = int(left)
            if right in month_var_map:
                cal_var = month_var_map[right]
                new_name = f"month_{mm:02d}_{cal_var}"
                rename_map[col] = new_name
```

```
# --- (B) FW variables: "FW0p10M-BIOMASS" -> "days_0.1_WPlant" (and same for
LAI)
fw_frac_map = {"FW0p10M": "0.1", "FW0p50M": "0.5", "FW0p90M": "0.9"}
fw_var_map = {"BIOMASS": "WPlant", "LAI": "LAI"}

for col in merged_op_try.columns:
    if col.startswith("FW") and "-" in col:
        left, right = col.split("-", 1)     # e.g. left="FW0p10M",
right="BIOMASS"
        if left in fw_frac_map and right in fw_var_map:
            frac = fw_frac_map[left]
            cal_var = fw_var_map[right]
            rename_map[col] = f"days_{frac}_{cal_var}"

# --- (C) NAN stats: "NANMAX-LAI" -> "max_LAI", etc.
nan_stat_map = {"NANMAX": "max", "NANMEAN": "mean", "NANSTD": "std"}
nan_var_map = {"BIOMASS": "WPlant", "LAI": "LAI"}

for col in merged_op_try.columns:
    if col.startswith(("NANMAX-", "NANMEAN-", "NANSTD-")) and "-" in col:
        left, right = col.split("-", 1)  # e.g. left="NANMAX", right="LAI"
        if left in nan_stat_map and right in nan_var_map:
            stat = nan_stat_map[left]
            cal_var = nan_var_map[right]
            rename_map[col] = f"{stat}_{cal_var}"

# Apply rename
merged_op_try = merged_op_try.rename(columns=rename_map)

print(f"Renamed {len(rename_map)} columns.")
# quick sanity check
print("Example renames:")
for k in list(rename_map.keys())[:10]:
    print(f"  {k} -> {rename_map[k]}")



#%%

# change of etimology for choerence with calibration
```

```
input_df_val = merged_op_try
X_names_val = input_df_val.columns.tolist()
X_val = input_df_val.to_numpy()

print("Validation predictors shape:", X_val.shape)
print("N. validation variables:", len(X_names_val))




# Yield conversion from fresh weight to dry matter
MOISTURE = 0.14
Y_val_op = (Y_val * (1.0 - MOISTURE)) / 1000.0  # DM t/ha



desc_basics_Y_val_op = pd.Series(Y_val_op, name="sorg_DM").describe()
desc_basics_Y_val_op["variance"] = desc_basics_Y_val_op["std"]**2
desc_basics_Y_val_op["|CV|"]                                                    =
abs(desc_basics_Y_val_op["std"]/desc_basics_Y_val_op["mean"])

sns.set(style="whitegrid")

plt.figure(figsize=(6,4))
sns.histplot(Y_val_op, bins=15, kde=True)

plt.xlabel("Yield (tons/ha)")
plt.ylabel("Frequency")

plt.tight_layout()
plt.show()



sns.set(style="whitegrid")
plt.figure(figsize=(4,6))
sns.boxplot(y=Y_val_op)

plt.ylabel("Yield (tons/ha)")

plt.tight_layout()
```

```
plt.show()



#%% Model validation (OPERATIVE)

# --- load artifacts ---
lr_model_op = joblib.load("lr_model_12vars_op.pkl")
rf_model_op = joblib.load("rf_model_12vars_op.pkl")
scaler_top50_op = joblib.load("scaler_top50_op.pkl")

vars_top50_sorted_op = joblib.load("vars_top50_sorted_op.pkl")
vars_12_op = joblib.load("vars_12_op.pkl")

# check similarities between variables
print("12 vars:", vars_12_op)

df_cal_12 = pd.DataFrame(X_train_op, columns=X_names_op)[vars_12_op]
df_val_12 = input_df_val[vars_12_op]

print("\nCAL (train) stats:")
stats_12_cal = df_cal_12.describe()
print(df_cal_12.describe().loc[["mean","std","min","max"]])

print("\nVAL stats:")
stats_12_val = df_val_12.describe()
print(df_val_12.describe().loc[["mean","std","min","max"]])



# --- build X_val for TOP50 (same columns, same order as training) ---
missing_top50 = [v for v in vars_top50_sorted_op if v not in
input_df_val.columns]
if missing_top50:
    raise KeyError(f"Validation is missing these TOP50 columns:
{missing_top50}")

X_val_top50 = input_df_val[vars_top50_sorted_op].to_numpy()
```

```
# --- standardize using TRAIN stats (scaler fit on training top50) ---
X_val_top50_std = scaler_top50_op.transform(X_val_top50)

# --- pick the 12 variables inside the top50 (same order as vars_12_op) ---
idx_12_in_top50 = [vars_top50_sorted_op.index(v) for v in vars_12_op]
X_val_12_std = X_val_top50_std[:, idx_12_in_top50]
X_val_12_raw = X_val_top50[:, idx_12_in_top50]   # per RF

# ============================================================
# REMOVE ROWS WITH NaN/Inf (ONLY based on the 12 variables)
# ============================================================
mask_ok_12 = np.isfinite(X_val_12_std).all(axis=1)  # True if no NaN/Inf in
the 12 std features

n_before = X_val_12_std.shape[0]
n_removed = int((~mask_ok_12).sum())
n_after = int(mask_ok_12.sum())

print("\nNaN check on 12 vars:")
print(f"Rows before: {n_before}")
print(f"Rows removed (NaN/Inf in 12 vars): {n_removed}")
print(f"Rows kept: {n_after}")

# apply mask to X and Y (IMPORTANT: keep alignment!)
X_val_12_std = X_val_12_std[mask_ok_12, :]
X_val_12_raw = X_val_12_raw[mask_ok_12, :]
Y_val_op_clean = np.array(Y_val_op)[mask_ok_12]




#%% Correlation between selected 12 vars and validation output

from scipy.stats import pearsonr, spearmanr

# check variables exist
missing_vars = [v for v in vars_12_op if v not in input_df_val.columns]
if missing_vars:
    raise KeyError(f"Missing validation variables: {missing_vars}")
```

```
X12_val_df = input_df_val[vars_12_op].copy()

# remove rows with NaN in predictors or output
mask_valid = (~X12_val_df.isna().any(axis=1)) & (~pd.isna(Y_val_op))

X12_val_clean = X12_val_df.loc[mask_valid]
Y_val_clean = Y_val_op[mask_valid]

print("Rows used for correlation:", len(Y_val_clean))

# compute correlations
rows = []
for v in vars_12_op:
    r_p, p_p = pearsonr(X12_val_clean[v], Y_val_clean)
    r_s, p_s = spearmanr(X12_val_clean[v], Y_val_clean)

    rows.append({
        "Variable": v,
        "Pearson": r_p,
        "p_Pearson": p_p,
        "Spearman": r_s,
        "p_Spearman": p_s,
        "Score": max(abs(r_p), abs(r_s))
    })

T_corr_val = (
    pd.DataFrame(rows)
      .sort_values("Score", ascending=False)
      .reset_index(drop=True)
)


# Tabella di correlazione SOLO per le 12 variabili selezionate (calibration)

T_corr_12_cal = (
    T_top50_op[T_top50_op["Variable"].isin(vars_12_op)]
        .set_index("Variable")
        .loc[vars_12_op]     # mantiene lo stesso ordine delle 12 variabili
```

```
        .reset_index()
)



#%% --- predictions ---
y_lr_pred_op = lr_model_op.predict(X_val_12_std)
y_rf_pred_op = rf_model_op.predict(X_val_12_raw)

# --- metrics vs observed validation output ---
r2_lr_val = r2_score(Y_val_op_clean, y_lr_pred_op)
rmse_lr_val = np.sqrt(mean_squared_error(Y_val_op_clean, y_lr_pred_op))
r_pearson_lr, _ = pearsonr(Y_val_op_clean, y_lr_pred_op)
r2_pearson_lr = r_pearson_lr**2
bias_lr = np.mean(Y_val_op_clean) - np.mean(y_lr_pred_op)

r2_rf_val = r2_score(Y_val_op_clean, y_rf_pred_op)
rmse_rf_val = np.sqrt(mean_squared_error(Y_val_op_clean, y_rf_pred_op))
r_pearson_rf, _ = pearsonr(Y_val_op_clean, y_rf_pred_op)
r2_pearson_rf = r_pearson_rf**2
bias_rf = np.mean(Y_val_op_clean) - np.mean(y_rf_pred_op)

# --- calibration metrics (already computed during training) ---
print("\n=== CALIBRATION PERFORMANCE ===")
print(f"LR (12 vars)  -> R2 = {r2_lr12_op:.3f}, RMSE = {rmse_lr12_op:.3f}, Pearson = {r_lr12_op:.3f}, Pearson2 = {r2p_lr12_op:.3f}, Bias = {bias_lr12_op:.3f}")
print(f"RF (12 vars)  -> R2 = {r2_12_op:.3f}, RMSE = {rmse_12_op:.3f}, Pearson = {r_rf12_op:.3f}, Pearson2 = {r2p_rf12_op:.3f}, Bias = {bias_rf12_op:.3f}")


print("\n=== VALIDATION PERFORMANCE ===")
print(f"LR (12 vars)  -> R2 = {r2_lr_val:.3f}, RMSE = {rmse_lr_val:.3f}, Pearson = {r_pearson_lr:.3f}, Pearson2 = {r2_pearson_lr:.3f}, Bias = {bias_lr:.3f}")
print(f"RF (12 vars)  -> R2 = {r2_rf_val:.3f}, RMSE = {rmse_rf_val:.3f}, Pearson = {r_pearson_rf:.3f}, Pearson2 = {r2_pearson_rf:.3f}, Bias = {bias_rf:.3f}")




#%% scatter observed vs predicted
plt.figure(figsize=(6, 6))
```

```
plt.scatter(Y_val_op_clean, y_rf_pred_op, alpha=0.5)

# bisettrice y = x
lims = [
    min(Y_val_op_clean.min(), y_rf_pred_op.min()),
    max(Y_val_op_clean.max(), y_rf_pred_op.max())
]
plt.plot(lims, lims)   # linea 1:1
plt.xlim(lims)
plt.ylim(lims)

plt.xlabel("Observed yield (t/ha)")
plt.ylabel("Predicted yield (t/ha)")
plt.title("Observed vs Predicted — RF validation")

plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()



# CARINO

plt.figure(figsize=(6, 6))

# scatter
plt.scatter(
    Y_val_op_clean,
    y_rf_pred_op,
    alpha=0.6,
    edgecolors='black',
    linewidths=0.4
)

# linea 1:1
lims = [
    min(Y_val_op_clean.min(), y_rf_pred_op.min()),
    max(Y_val_op_clean.max(), y_rf_pred_op.max())
]
plt.plot(lims, lims, linestyle='--', linewidth=1.5)
```

```
plt.xlim(lims)
plt.ylim(lims)

plt.xlabel("Observed yield (t/ha)")
plt.ylabel("Predicted yield (t/ha)")
plt.title("Observed vs Predicted Yield — Random Forest (Validation)")

# griglia leggera
plt.grid(True, linestyle='--', alpha=0.2)

# aspetto quadrato perfetto
plt.gca().set_aspect('equal', adjustable='box')

plt.tight_layout()
plt.show()

# %% --- split train/test sui dati osservati ---
Xv_train, Xv_test, Yv_train, Yv_test = train_test_split(
    X_val_12_raw,
    Y_val_op_clean,
    test_size=0.2,
    random_state=42
)

# --- RF ---
start = time.time()
r2_val_rf_obs,    rmse_val_rf_obs,    r_rf_obs,    r2p_rf_obs,    bias_rf_obs,
imp_val_rf_obs, rf_val_obs = run_random_forest(
    Xv_train, Xv_test, Yv_train, Yv_test,
    feature_names=vars_12_op
)

yieldv_pred = rf_val_obs.predict(Xv_test)

print(f"Time {time.time() - start:.2f} s")
print(f"RF   -   OBSERVATIONS   -->   R2   =   {r2_val_rf_obs:.3f},   RMSE   =
{rmse_val_rf_obs:.3f}, Pearson = {r_rf_obs:.3f}, Pearson2 = {r2p_rf_obs:.3f},
Bias = {bias_rf_obs:.3f}")
print(imp_val_rf_obs)
```

```
del start
```

```
# %% EUREQA
# 1) nomi variabili (ordine = colonne di X_top50_*_op)
print(sorted_vars_op)
# 2) scaler fit su TRAIN, applicato a TRAIN e TEST
scaler_eur_op = StandardScaler()
X_top50_train_std_op = scaler_eur_op.fit_transform(X_top50_train_op)
X_top50_test_std_op  = scaler_eur_op.transform(X_top50_test_op)

# 3) array unico (train + test)
X_top50_all_std_op = np.vstack([X_top50_train_std_op, X_top50_test_std_op])

print("X_top50_all_std_op shape:", X_top50_all_std_op.shape)
print("N. top50 vars:", len(sorted_vars_op))

#%% ===== 1) helper: estrai una colonna per nome dalla matrice X (ordine =
sorted_vars_op)
def col(X, name, var_order):
    idx = var_order.index(name)
    return X[:, idx]

#%% ===== 2) Eureqa model (ATTENZIONE: assumo che sia stato fit su variabili
STANDARDIZZATE)
def eureqa_predict(X, var_order):
    max_WPlant   = col(X, "max_WPlant", var_order)
    month_06_ET  = col(X, "month_06_ET", var_order)
    month_05_LAI = col(X, "month_05_LAI", var_order)
    std_Wplant   = col(X, "std_WPlant", var_order)

    y_pred = (
```

```
        0.940862096666509
        + 2.70643300931012 * max_WPlant
        + 0.175655707506274 * month_06_ET
        - 0.140550329531939 * month_05_LAI
        - 2.45469494364294  * std_Wplant
    )
    return y_pred

#%% ===== 3) metriche richieste
def regression_metrics(y_true, y_pred):
    y_true = np.asarray(y_true).ravel()
    y_pred = np.asarray(y_pred).ravel()

    r2 = r2_score(y_true, y_pred)
    rmse = np.sqrt(mean_squared_error(y_true, y_pred))
    r, _ = pearsonr(y_true, y_pred)
    r2_pearson = r**2
    mean_bias = np.mean(y_pred - y_true)   # bias positivo = sovrastima media

    return {
        "R2": r2,
        "RMSE": rmse,
        "Pearson": r,
        "Pearson^2": r2_pearson,
        "Mean Bias": mean_bias
    }


#%% --- variables used by Eureqa model ---
vars_eur4 = ["max_WPlant", "month_06_ET", "month_05_LAI", "std_WPlant"]

# --- find their indices inside the top50 (order = sorted_vars_op) ---
idx_eur4_in_top50 = [sorted_vars_op.index(v) for v in vars_eur4]

# --- extract only those 4 columns ---
X_val_eur4_std = X_val_top50_std[:, idx_eur4_in_top50]

# ============================================================
# REMOVE ROWS WITH NaN/Inf (ONLY based on the 4 Eureqa variables)
```

```
# ============================================================
mask_ok_4 = np.isfinite(X_val_eur4_std).all(axis=1)

n_before = X_val_eur4_std.shape[0]
n_removed = int((~mask_ok_4).sum())
n_after = int(mask_ok_4.sum())

print("\nNaN check on Eureqa 4 vars:")
print(f"Rows before: {n_before}")
print(f"Rows removed (NaN/Inf in Eureqa 4 vars): {n_removed}")
print(f"Rows kept: {n_after}")

# apply mask to X and Y (IMPORTANT: keep alignment!)
X_val_eur4_std = X_val_eur4_std[mask_ok_4, :]
Y_val_eur_clean = np.asarray(Y_val_op).ravel()[mask_ok_4]



# ============================================================
#%% (A) TEST: X_top50_test_op vs Y_test
# ============================================================

# Se NON hai già X_top50_test_std_op, crea così (coerente col tuo workflow):
# X_top50_test_std_op = scaler_eur_op.transform(X_top50_test_op)

y_test_pred = eureqa_predict(X_top50_test_std_op, sorted_vars_op)
metrics_test = regression_metrics(Y_test, y_test_pred)

print("=== TEST (Eureqa) ===")
for k, v in metrics_test.items():
    print(f"{k:10s}: {v:.4f}")

# ============================================================
#%% (B) VALIDATION: X_val_top50 vs Y_val_clean
# ============================================================

# Eureqa model using a 4-column matrix in this exact order:
# ["max_WPlant", "month_06_ET", "month_05_LAI", "std_Wplant"]
def eureqa_predict_4cols(X4):
    max_WPlant   = X4[:, 0]
```

```
    month_06_ET  = X4[:, 1]
    month_05_LAI = X4[:, 2]
    std_Wplant   = X4[:, 3]

    return (
        0.940862096666509
        + 2.70643300931012 * max_WPlant
        + 0.175655707506274 * month_06_ET
        - 0.140550329531939 * month_05_LAI
        - 2.45469494364294  * std_Wplant
    )

# ============================================================
# (B) VALIDATION: Eureqa on 4 vars (std) vs Y_val_eur_clean
# ============================================================
y_val_pred = eureqa_predict_4cols(X_val_eur4_std)
metrics_val = regression_metrics(Y_val_eur_clean, y_val_pred)

print("\n=== VALIDATION (Eureqa) ===")
for k, v in metrics_val.items():
    print(f"{k:10s}: {v:.4f}")
```

B.2. "T_top50": dataframe of 50 most correlated variables with the output (*potential*)

	Variable	Pearson	p_Pearson	Spearman	p_Spear-man	Score
0	month_05_NP	0.666	0	0.568	0	0.666
1	month_05_NPlant	0.550	0	0.565	0	0.565
2	max_NPlant	0.536	0	0.560	0	0.560
3	month_04_NP	0.555	0	0.498	0	0.555
4	month_03_NPlant	0.523	0	0.549	0	0.549
5	std_NPlant	0.511	0	0.549	0	0.549
6	month_04_NPlant	0.526	0	0.545	0	0.545
7	days_0.8_WPlant	-0.428	0	-0.473	0	0.473

8	max_WPlant	0.398	0	0.458	0	0.458
9	month_02_NPlant	0.392	0	0.449	0	0.449
10	month_06_T	0.321	0	0.448	0	0.448
11	month_06_ET	0.316	0	0.404	0	0.404
12	month_05_WPlant	0.324	0	0.403	0	0.403
13	days_0.2_WPlant	-0.390	0	-0.250	0	0.390
14	doy_max_WPlant	0.369	0	0.363	0	0.369
15	days_0.1_WPlant	-0.365	0	-0.226	0	0.365
16	mean_NPlant	0.303	0	0.364	0	0.364
17	days_0.5_NPlant	-0.364	0	-0.221	0	0.364
18	days_0.9_WPlant	-0.232	0	-0.360	0	0.360
19	month_11_ET	-0.357	0	-0.315	0	0.357
20	month_12_ET	-0.297	0	-0.353	0	0.353
21	days_0.5_WPlant	-0.350	0	-0.297	0	0.350
22	month_06_NP	0.343	0	0.199	0	0.343
23	month_06_NPlant	0.334	0	0.338	0	0.338
24	month_01_NPlant	0.229	0	0.326	0	0.326
25	days_0.9_LAI	-0.325	0	-0.301	0	0.325
26	month_06_ET_0	-0.319	0	-0.291	0	0.319
27	std_WPlant	0.230	0	0.318	0	0.318
28	days_0.1_NPlant	-0.309	0	-0.227	0	0.309
29	month_01_LAI	-0.309	0	-0.079	########	0.309
30	month_10_ET	-0.274	0	-0.306	0	0.306
31	days_0.2_NPlant	-0.304	0	-0.201	0	0.304
32	month_04_LAI	0.275	0	0.303	0	0.303
33	month_02_LAI	-0.300	0	-0.001	0.489923	0.300
34	month_06_T_0	-0.282	0	-0.300	0	0.300
35	month_12_T	-0.267	0	-0.289	0	0.289
36	month_02_NP	0.278	0	0.285	0	0.285
37	month_12_LAI	-0.280	0	-0.149	0	0.280
38	month_03_NP	0.270	0	0.276	0	0.276
39	doy_max_LAI	-0.275	0	-0.158	0	0.275
40	month_05_LAI	0.275	0	0.266	0	0.275
41	month_11_T	-0.273	0	-0.241	0	0.273
42	days_0.5_LAI	-0.268	0	-0.137	0	0.268
43	month_06_WPlant	0.205	0	0.260	0	0.260
44	month_06_ET_ref	-0.251	0	-0.119	0	0.251

45	month_10_NP	-0.248	0	-0.225	0	0.248
46	month_11_LAI	-0.241	0	-0.163	0	0.241
47	days_0.2_LAI	-0.240	0	-0.147	0	0.240
48	month_05_T	0.139	0	0.240	0	0.240
49	days_0.1_LAI	-0.238	0	-0.168	0	0.238

"""

B.3. "T_top50": dataframe of 50 most correlated variables with the output (*operational*)

	Variable	Pearson	p_Pearson	Spearman	p_Spear-man	Score
0	max_WPlant	0.398	0	0.458	0	0.458
1	month_06_T	0.321	0	0.448	0	0.448
2	month_06_ET	0.316	0	0.404	0	0.404
3	month_05_WPlant	0.324	0	0.403	0	0.403
4	days_0.1_WPlant	-0.365	0	-0.226	0	0.365
5	days_0.9_WPlant	-0.232	0	-0.360	0	0.360
6	month_11_ET	-0.357	0	-0.315	0	0.357
7	month_12_ET	-0.297	0	-0.353	0	0.353
8	days_0.5_WPlant	-0.350	0	-0.297	0	0.350
9	days_0.9_LAI	-0.325	0	-0.301	0	0.325
10	month_06_ET_0	-0.319	0	-0.291	0	0.319
11	std_WPlant	0.230	0	0.318	0	0.318
12	month_01_LAI	-0.309	0	-0.079	#########	0.309
13	month_10_ET	-0.274	0	-0.306	0	0.306
14	month_04_LAI	0.275	0	0.303	0	0.303
15	month_02_LAI	-0.300	0	-0.001	0.489923	0.300
16	month_06_T_0	-0.282	0	-0.300	0	0.300
17	month_12_T	-0.267	0	-0.289	0	0.289
18	month_12_LAI	-0.280	0	-0.149	0	0.280
19	month_05_LAI	0.275	0	0.266	0	0.275
20	month_11_T	-0.273	0	-0.241	0	0.273
21	days_0.5_LAI	-0.268	0	-0.137	0	0.268
22	month_06_WPlant	0.205	0	0.260	0	0.260
23	month_11_LAI	-0.241	0	-0.163	0	0.241
24	month_05_T	0.139	0	0.240	0	0.240
25	days_0.1_LAI	-0.238	0	-0.168	0	0.238
26	mean_LAI	-0.234	0	-0.105	0	0.234
27	month_05_ET	0.149	0	0.225	0	0.225
28	month_03_T	-0.188	0	-0.223	0	0.223

29	month_01_ET	-0.205	0	-0.216	0	0.216
30	month_10_T	-0.203	0	-0.188	0	0.203
31	max_LAI	0.066	########	0.201	0	0.201
32	month_03_ET_0	0.161	0	0.199	0	0.199
33	month_10_LAI	-0.189	0	-0.177	0	0.189
34	month_04_WPlant	0.099	0	0.181	0	0.181
35	month_03_ET	-0.160	0	-0.180	0	0.180
36	month_01_T	-0.177	0	-0.106	0	0.177
37	month_10_ET_0	-0.168	0	-0.085	0	0.168
38	std_LAI	-0.167	0	-0.061	########	0.167
39	month_09_ET_0	0.019	1.37E-18	0.160	0	0.160
40	month_08_ET_0	-0.123	0	-0.149	0	0.149
41	month_05_T_0	-0.074	########	-0.140	0	0.140
42	month_08_T_0	-0.090	0	-0.139	0	0.139
43	month_01_ET_0	-0.139	0	-0.073	########	0.139
44	month_08_ET	0.055	########	0.139	0	0.139
45	month_04_T	-0.136	0	-0.135	0	0.136
46	month_12_ET_0	-0.129	0	-0.022	9.76E-26	0.129
47	month_02_T_0	0.005	0.03063	0.125	0	0.125
48	month_04_ET	-0.124	0	-0.120	0	0.124
49	month_07_T_0	-0.112	0	-0.077	########	0.112

List of Figures

List of Tables

List of Equations

Acknowledgments

I would like to thank Prof. Chiara Corbari for her guidance and support throughout this work.

I also thank Héctor Nieto Solána for his help and valuable advice during the development of this thesis.

Last, but not least, I would like to thank everyone else: my family and friends.