# Better Emulators

TSladen

2024-08-28

## Introduction

This project aims to develop multiple emulators using different methods for the Joint UK Land Environment Simulator (JULES). Where simulators are mathematical computer-based models used to simulate earth system processes such as the land surface whilst emulators are used to provide predictions based on given simulator inputs (Salter and Williamson, 2016).

To which JULES provides high resolution simulations of various aspects of the land surface such as for carbon, water and energy fluxes (Clark et al, 2011) where the land surface is an important part of the earth system which directly impacts both humans and other terrestrial organisms (Fisher and Koven, 2020). This highlights the importance of understanding land surface – climate interactions where changes in climate influence ecosystem processes which further influence climate resulting in a feedback loop (Chu et al, 2016) which does not just affect regional areas but instead all global regions.

Therefore, the data used for this project focused on carbon-based aspects of the land surface with terrestrial ecosystems acting as important factors in terms of fluxes where they can release/absorb greenhouse gases into/from the atmosphere whilst also influencing energy and water fluxes (Heimann and Reichstein, 2008). Where terrestrial ecosystems obtain much of their carbon from photosynthesis whilst losing carbon mainly through respiration (Heimann and Reichstein, 2008). To which global carbon is represented through various features such as gross primary production (GPP) which represents the total amount of carbon stored by plants within a specified spatial area under a set time-period (Roxburgh et al, 2005). However, anthropogenic changes in land use over time such as through increases in agriculture and deforestation have resulted in changes in the land surface – atmosphere carbon fluxes typically with more carbon being released to the atmosphere (Meiyappan and Jain, 2012).

Due to this, land surface models such as JULES have been developed to allow us to understand how these changes may impact future climate highlighting the need for accurate simulations. Furthermore, there are high computational costs associated with these simulations meaning they cannot be run repeatedly therefore, there is a drive to use cheaper statistical emulators to instead make predictions based on their respective given simulator inputs (Baker et al, 2022). However, many different emulation methods are available leading to the question of what emulation method to use. This question leads to the overall goal of this project which is to determine which of the emulator methods used provided the best results for the JULES data. To determine this several aims have been laid out: Model accuracy: Asses how well the different emulation methods predict observations and the associated error.

Model precision: Evaluate the certainty of the predictions made through each method, indicated by the range width of possible values the models have determined.

Computational cost: Measure the time taken to tune and train each methods' models, where a preference would be with quicker run times.

Therein the culmination of the information generated from these aims will allow for an informed decision when deciding which emulation method is most suitable for emulating the JULES data.

## Literature Review

To provide context and motivation for this project a study by McNeall et al, 2024 is used to explain where the data originates and how it was created to which this project builds upon the emulation aspect of this study. Wherein this study focused on restraining the input space for the carbon cycle within JULES. The data used for this project is also the same as from this study where their experiment consisted of two waves of JULES ensembles wave 0 for exploratory purposes where the setup for wave 1 was then based on the results of wave 0 (McNeal et al, 2024).

The simulations were based on land-use and atmospheric carbon changes with the simulation timeframe being 1850 to 2014 where post industrial observations were created by modifying the 1850-69 climate whilst maintaining the 1850 CO2 levels (McNeall et al, 2024). Additionally, each ensemble instance used different input value combinations to which there was a total of 32 input parameters with these parameters being chosen based on potential importance in influencing land surface processes (McNeall et al, 2024). These parameters were changed randomly using the maximin Latin hypercube method wherein this allowed for many unique settings to create a range of results to adequately train an emulator (McNeall et al, 2024). The purpose of this was to be able to train Gaussian process emulators for different output parameters to explore many relationships as if a larger ensemble was used (McNeall et al, 2024).

The wave 0 ensembles, which were used for this project, possessed 499 iterations for the purpose of having at least 10 per input parameter whilst having enough members for a train/test data a split (McNeall et al, 2024). The purpose of wave 0 was to find the limits of the input space this way future ensembles will be based on a range that has already been tested (McNeall et al, 2024).

This study snippet highlights the initial expectations we can have for the performance of each emulation method where, based on wave 0 ensembles we would not be expecting perfect emulator performance as many of the ensemble members may not be considered feasible in the real world. However, what this allows for is testing the limits of each method and how well it performs with non-perfect data along with investigating any potential biases that arise from predictions.

Also, to provide context behind the use of principal component analysis (PCA) for emulators a study snippet by Higdon et al, 2008 is examined. Where the initial goal is to make emulators for a complex implosion simulator but this comes with a challenge of a high dimensional output (Higdon et al, 2008). Thereby, they implement a $p_\eta$-dimensional basis representation for their output which simplifies the modelling by making the output a linear combination of basis vectors defined by the following equation (Higdon et al, 2008):

$$\eta(x,t) = \sum_{i=1}^{p} k_i w_i(x,t) + \epsilon, (x,t) \in [0,1]^{px+pt}$$

Where $\eta(x,t)$ is the simulator output, $k_i$ are the principal components, $w_i(x,t)$ are the principal component coefficients modeled by the Gaussian processes and $\epsilon$ is an error term where overall, this reduces the complexity of the modelling process (Higdon et al, 2008). Where instead of training the emulators to predict values for each feature (in my case for each year) the emulator is trained to predict the coefficients of the principal components to which these represent the underlying patterns of the data with respect to changes in the input values (Higdon et al, 2008). Essentially this allows the emulator to learn to predict patterns rather than the full output space directly.

This study highlights the application of using pca to reduce the complexity and improve the efficiency of modelling a high dimensional output where this application could be a useful approach to apply to emulating JULES which also produces high dimensional outputs. Where due to the output features of my data representing years it can be expected their would be high collinearity between features meaning pca would be applicable.

Additionally, a study by Salter et al, 2019 focused on uncertainty quantification for computer models with an application to climate science using calibration optimal bases. Where it was stated for the purpose of calibrating a representation of a high dimensional output there are two main points: representing the observations sufficiently and retaining enough variance for accurate emulations (Salter et al, 2019). Where to obtain sufficient representation of the true observations the reconstruction error should be minimized, and to do this it was suggested to retain at least 95% of the variance, where the reduction was accomplished by singular value decomposition (SVD, as opposed to PCA like in my own project) (Salter et al, 2019). Furthermore they highlight the use of an optimal rotation algorithm which to summarise, effectively determines the optimal number of basis vectors (principal components) by minimizing the reconstruction error to ensure accurate predictions by using a sufficient amount of vectors to represent the majority of the variance (Salter et al, 2019).

This study highlighted the importance of when applying dimensionality reduction techniques that sufficient variance should be retained to ensure the reconstructed data is as close to the original as possible to ensure the emulators are able to produce accurate outputs.

Furthermore an initial search was conducted on previous attempts at emulating JULES. To this end a study by Baker et al, 2022 applied sparse Gaussian processes to emulating gross primary productivity (GPP) using JULES data with an overall goal of producing an emulation framework for land surface models using sparse Gaussian processes. Where they state a Gaussian process emulator can be considered as (Baker et al, 2022):

$$y(\theta) \sim GP(m(\theta), k(\theta, \theta'))$$

Where y is the simulator output, $\theta$ are the inputs, m is the mean function and k is the covariance function (Baker et al, 2022).

Therefore, to assess the accuracy of their emulator 10% of the data points were withheld for use as testing data where for the five emulators made they used 1000 predictions which were randomly chosen points from the test data to which they used this to obtain two standard deviation intervals (Baker et al, 2022). Where they stated the two intervals should correspond to the 95.4% confidence interval and therefore about 95.4% of the true observations should fall within the standard deviation interval of the emulators predictions (Baker et al, 2022). Where the obtained accuracies were 94.8%, 95.6%, 95.6%m 95.4% and 95% implying good performance from all the emulators (Baker et al, 2022).

Additionally, they highlighted the use of GP emulators for preliminary sensitivity analysis if the corvariance function is used an relevance determination kernel where the size of the lengthscale values determines feature importance where a smaller lengthscale suggests greater importance (Baker et al, 2022). In this regard the feature air surface temperature was found to be the most important input (Baker et al, 2022).

Therefore, to understand what other methods have been tried before and thus what may be relevant for use in the context of JULES various past research papers about environmental emulations and applications of machine learning to environmental data have been reviewed.

For example, a paper produced by Pal and Sharma 2021 reviewed various machine learning applications to land surface modelling and how machine learning has allowed further progress in data-based earth science research (Pal and Sharma, 2021). To this end it was stated that machine learning algorithms typically fall into two categories: traditional methods and deep learning where examples of traditional methods include random forests and classification/regression trees whilst deep learning typically revolves around various types of neural networks (Pal and Sharma, 2021). Their aims for this review were to highlight areas of improvement in modelling, discuss the most important machine learning methods and suggest future directions/improvements (Pal and Sharma, 2021).

Within the context of evapotranspiration, a reviewed paper by Pan et al 2020, attempted to predict global terrestrial evapotranspiration using approaches in remote sensing and machine learning methods such as a random forest (Pan et al, 2020, Pal and Sharma, 2021). Different methods of evaluating evapotranspiration were compared these being land surface models, machine learning methods and physics models were some examples for each method include: JULES for the land surface model, random forest for machine learning and MODIS for physics based models (Pan et al, 2020).

They found all methods detected an increase in evapotranspiration globally however they noted large uncertainties in some areas such as land surface models having high uncertainty in tropical regions whilst other methods had greater uncertainty within arid regions (Pan et al, 2020).

However, it was concluded that the machine learning methods yielded results comparable to direct remote sensing highlighting their use of making accurate predictions based on land surface model data (Pan et al, 2020, Pal and Sharma, 2021).

Furthermore, their applications to land surface models were also described where it was suggested random forests are typically used for regression-based problems such as continuous prediction where it was stated random forests are found to be consistent and adaptive whilst being able to handle high dimensionality datasets (Pal and Sharma, 2021).

Based on this review I determined a random forest would be applicable to my projects context which is focussed on regression-based emulators for a dataset with high dimensionality. However, to better understand how random forests are applied to environmental problems further papers were investigated.

Such as a from this study by Yang et al 2017, which focused on downscaling land surface temperature (LST) data in arid regions using multiple remote sensing indices with random forests where downscaling involves predicting continuous temperature values under a higher resolution (Yang et al, 2017). The downscaling used data from MODIS LST where the resolution was downscaled from 1km to 500m where specific indices were selected and the regression models between predictors and LST were used to enhance the LST resolution with models then being evaluated using the r-squared, bias and RMSE (Yang et al, 2017).

The results of this study found the downscaling to 500m produced reliable and more detailed information compared to the original 1km version produced by MOD11 with their approach achieving an r-squared of [0.967] and an associated RMSE of [1.537] however it was also found the random forest tended to underestimate where out of the six sites tested four of them had predictions bellow the true value (Yang et al, 2017).

Although this study uses random forest regression under a different context to emulation it provides evidence in how random forests can be used to make predictions for environmental data and the associated effectiveness of the method with the results suggesting the method is worth testing for my project to determine their effectiveness at emulating land surface models.

Furthermore, a paper by Chen and Guestrin 2016, described the machine learning method XGBoost. Where the XGBoost method uses tree ensembles to determine the best outcomes for regression and classification problems whilst also containing imbedded techniques to reduce overfitting such as shrinkage and column subsampling (Chen and Guestrin, 2016). Where they also summarise that it can solve real world problems with few resources (Chen and Guestrin, 2016).

However, to understand its use within the context of environmental data other papers were investigated. For example, a paper by Keetz et al 2024 which investigated using different emulation methods following the calibrate, emulate, sample framework (CES) to determine the best set of values for the vegetation demographics model Functionally Assembled Terrestrial Ecosystem Simulator (FATES) within the Community Land Model 5 (CLM) (Keetz et al, 2024). Where an initial estimate of the posterior distribution was calculated with an emulator then being trained on the observation mismatches of the model to predict observations for unseen inputs where the emulator eventually replaces the simulator for faster computation of posterior sampling (Keetz et al, 2024). Where, this was undertaken using evapotranspiration and gross primary production data (Keetz et al, 2024).

However, they found in general the Gaussian process emulator performed better than the XGBoost regarding predicted vs observed log likelihood where the Gaussian process produced an r-squared of [0.9991] with an RMSE of [2.43] compared to the XGBoost with an r-squared of [0.9828] and an RMSE of [10.77] (Keetz et al, 2024). Where it was concluded that since the Gaussian process model successfully extended the model mismatches in the parameter region of interest it was determined as a suitable surrogate to replace the full simulation for sampling the posterior (Keetz et al, 2024).

From this paper although the XGBoost model may not have performed as well as the Gaussian process it highlights how this method can be used within the context of making predictions based on land surface simulator outputs. Where a similarity to my project lies in the goal of overall replacing a simulator with an emulator wherein this relates to my aim of producing accurate predictions given a set of input values.

Furthermore, a study by Jia et al, 2021 assessed the performance of differing machine learning methods to estimate soil moisture content. The study used 4 different methods XGBoost, random forest, artificial neural networks and support vector machines where a pre-classification strategy was also implemented which involved re-sampling and sub-modelling to reduce biases caused by different land types thereby increasing estimation accuracy (Jia et al, 2021).

The study uses satellite data from the CYGNSS collection of satellites which collects various types of meta data which overall were used to calculate surface reflectivity and SMAP which measures information on surface soil water content (Jia et al, 2021). The land types investigated include several forest types such as evergreen and deciduous alongside open grasslands/Savannas as well as croplands (Jia et al, 2021).

The estimations for the different methods were then evaluated such as through the RMSE for estimations using 10-fold cross validation (Jia et al, 2021). The results produced the following RMSE values XGBoost [0.052(cm3/cm3], random forest [0.060(cm3/cm3)], artificial neural network [0.063(cm3/cm3] and support vector machine [0.068cm3/cm3] (Jia et al, 2021). Where these results indicate that the XGBoost method had the best estimations for soil moisture followed by the random forest whilst the addition of the pre-classification strategy also suggested improvements XGBoost no pre-class [0.064(cm3/cm3)] (Jia et al, 2021).

This study also highlights the use of XGBoost for regression based environmental problems implying its potential applicability to my project albeit under a different context to emulation.

Furthermore, a Gaussian process emulator is being used as a baseline due to their previous applications to JULES such as within McNeall et al, 2024s study. However, additional studies were investigated to understand their performance in the context of environmental science.

For example, a study by Owen and Liuzzo, 2019 investigated using Gaussian processes to emulate a hydrological model on the catchment scale. Where the study area was based in southwest England for the river Frome within the East Stoke basin which focused on 4 land use input variables (woodland, arable, pasture and urban) with river flow as the output (Owen and Liuzzo, 2019).

Their emulator was found to be mostly successful where the emulator was able to reproduce the time pattern for daily and monthly average flow where the NS coefficients were 0.64 and 0.70 respectively where these values represent good overall performance (Owen and Liuzzo, 2019). Additionally, they compared the computational time of using the original model compared to the emulator with the model taking an average of 8.768 seconds to complete and the emulator 2.182 seconds suggesting a 4x decrease in computational time (Owen and Liuzzo, 2019).

This study showcases the performance of Gaussian processes for emulation in the context of environmental data and how they can be applied to environmental issues. Additionally, the use of computational time comparisons also showcases why we would want to use emulators to make predictions as opposed to running simulators or full models albeit in the context of their study the computational times were inherently low due to the simplicity of the original model compared to simulators such as JULES. However, their inclusion of computational time highlights its importance as factor for choosing emulators with their study comparing the original model and emulator whereas this project instead will focus on comparing computational times between emulation methods.

Additionally, a study by Donnelly et al, 2022 applied the use of Gaussian processes to hydrology which is an aspect of the water cycle part of land surface where this study focussed on examining the appropriateness of Gaussian processes to emulate a flood model.

The modelling framework created in the study was centred on urban flood modelling in Tadcaster UK with the model implementing topographic and inundation data (Donnelly et al, 2022). The flood simulator model (LISFLOOD-FP) was setup using a high-resolution digital elevation map with a resolution of 2 meters where natural and urban features were included to accurately represent how water would behave in an urban area (Donnelly et al, 2022). Boundary conditions were determined by sampling historic extreme peak discharges as the focus was on modelling extreme climatic events (Donnelly et al, 2022). Once the topographic data and boundary conditions were set LISFLOOD-FP simulated outcomes for the provided hydrographs where discharge values at set times were used to train the multi-output emulator (Donnelly et al, 2022).

To assess performance the Gaussian process emulator was assessed against a convolutional neural network where in cross validation performance the Gaussian process produced an RMSE of 0.209 and the neural network 0.232 suggesting an improvement (Donnelly et al, 2022). Additionally, computational times were also compared where the neural network had a training time of 600 seconds and a run-time of 80 seconds whilst the Gaussian process had a training time of 226 seconds and a run-time of 1 second comparatively the simulator had a run-time of 9,900 seconds implying that the Gaussian process can be applicable to emulating flood models (Donnelly et al, 2022).

Overall, this paper highlights the applications of a Gaussian process model to land surface processes alongside its performance and its potential suitability for application to a different component of the land surface, where in the case of this project this is for the carbon cycle. Additionally, the method comparison of computational time also validates my reasoning to showcase computational cost between methods as I view this as an important consideration when choosing an emulation method.

To summarise, the literature highlights applications of regression-based machine learning methods to various aspects of the environment including the land surface. Where many studies have compared different methods for the same task to understand the applicable of certain methods for certain problems. Additionally other information such as computational time is also presented in some cases providing evidence for the desire to reduce the computational cost of modelling complex systems. Furthermore, the study by McNeall et al, 2024 provides the context behind the data used and its aim to follow upon the original experiment with a focus on the emulation aspect of JULES.

## Methodology

Thereby, to undertake this project a total of three machine learning methods have been used these are as follows: random forests, Gaussian processes and extreme gradient boosting machines (XGBoost). With all these methods being used under the R programming environment (R Core team, 2024) where the following packages have been used: randomForest (Liaw and Wiener, 2002) for random forest models, DiceKriging (Roustant et al, 2012) for Gaussian process models and XGBoost (Chen et al, 2024) for XGBoost. Where within the literature these methods have been highlighted as suitable to apply for regression-based issues.

Furthermore, the first action taken was to reduce the dimensionality of the data. The purpose of this was to reduce computational cost and to extract only the most important features. To this end principal component analysis (pca) was used as most of the data follows linear patterns wherein pca is a linear dimensionality reduction technique. However, two datasets were highly non-linear those being NBP and FLuc thus, the cumulative versions were calculated to better linearise their patterns. The reason for this is due to pca splitting data based on linear combinations of the available features meaning it will be more effective when the data follows linear patterns. Although other methods of reduction exist for non-linear data such as kernel pca I decided for consistency it would be more suitable to be able to apply the same reduction method to all datasets. Additionally, the number of principal components retained was chosen based on how many were required to retain 99.9% of the variance where this was needed to allow for reduced reconstruction error when returning the data to the original space/scaling. The number of principal components needed for 99.9% of variance being retained for each dataset are outlined in table 1.

Table 1 Number of PCs used for each dataset

| DataSet | Number.of.Principal.Compents |
|---|---:|
| BareSoilFrac | 2 |
| C3Plants | 2 |
| C4Plants | 2 |
| CSoil | 1 |
| CVeg | 1 |
| fHarvest | 2 |
| CumFLuc | 5 |
| Lai | 1 |
| CumNBP | 5 |
| NPP | 2 |
| Rh_land_sum | 2 |
| ShrubFrac | 2 |
| TreeFrac | 1 |

To establish a baseline method a Gaussian process emulator was used. This is because they are often used to build emulators particularly within climate science (Baker et al, 2022). Where they are useful in this application as they are flexible and can fit smooth functions that account for complex interactions (Golding and Purse, 2016). This is particularly important for this project where the interactions within land surface systems are complex. To tune the Gaussian process Bayesian optimisation was used which is an efficient way to test a range of hyperparameter values by creating simpler models to maximise a score. In this case the score being maximised was RMSE where specifically the negative RMSE was used as the maximisation processes attempts to find the largest value but in the case of RMSE smaller values are better. Thereby when maximising a negative the process will attempt to find the combination of hyperparameters that result in the score closest to 0.

Additionally, the random forest has been chosen as a method due to their adaptability and ability to handle high dimensional data (Pal and Sharma, 2021). This is required as the adaptability was important in being able to apply the random forest method to multiple output features with many such features also exhibiting different observation scaling and patterns. Due to the dimensionality of the data used it was also important the model would be able to handle many dimensions without significant detriments to computational cost. Additionally, the results of Yang et al 2017s study also support the random forests capability of handling continuous data for regression-based problems. To tune the hyperparameters of the random forest a grid search was performed for components such as mtry, which represents how many inputs are tested at each node within the decision tree. Where grid searches allow for the testing of a range of hyperparameter value combinations thereby increasing the probability of finding optimal hyperparameter values the model can use.

Furthermore, the XGBoost method was chosen due to its inbuilt techniques to reduce overfitting where similarly to the random forest it is adaptable and suitable for large datasets (Chen and Guestrin, 2016). As with the random forest a grid search was used to tune the hyperparameters such as for max_depth which represents tree complexity i.e. how many decisions are being made per tree. Where again the grid search would allow for many combinations of hyperparameter values to be tested. Additionally, unlike Bayesian optimisation grid searches are simpler to setup albeit less efficient at exploring a given hyperparameter space.

Whereas, to calculate computational cost which within this project is represented as computational time the system.time function within R was used. In this regard system.time provides up to 3 values where the value of importance was elapsed time which represents how long a process took to complete. This value is measured in seconds therefore to convert it to minutes the values were divided by 60. Additionally, due to the use of looping to run multiple emulators more efficiently system.time was used within the loop as opposed to the loop being within system.time. This was because the aim was acquiring/storing the computational time for all datasets as opposed to just the overall time for each method. Where the overall times can instead be easily calculated by simply combining the individual dataset run times for each method.

## Data and Initial Analysis

As mentioned previously the data for this project comes from the JULES-ES-1.0 wave 0 simulations produced in McNeall et al, 2024s study. Where again the dataset's purpose was to fully explore the JULES input parameter space to create a range of ensemble members whether they would be realistically possible or not. However, to this end any ensemble members corresponding to zero-carbon runs were omitted with all other ensemble members being retained. The dimensionality of the data was as follows with input dimensions of [499x32] and output dimensions of [13[499x164]]. The 164 output features each represented a year with the overall timeframe being 1850-2014. Additionally, the 13 output datasets used corresponded to aspects of the carbon cycle these are as follows: BareSoilFrac, C3Plants, C4Plants, CSoil, CVeg, FHarvest, FLuc, Lai, NBP, NPP, RhLndSum, ShrubFrac, TreeFrac. A brief description of the output variables are presented in Table 2. Whereas, the 32 input features corresponded to factors which were thought to influence the carbon cycle, as this project does not focus on the input features used further information on how these features were selected can be found in McNeall et al, 2024s study whilst the parameters themselves and a brief description are presented in Table 3.

Table 2 Output Variable Descriptions

| Output_Variable | Description | Reference |
|---|---|---|
| BareSoilFrac | Proportion of land covered by bare soil | CEDA, 2024 |
| C3Plants | Proportion of land covered by C3 photosynthetic pathway plants | CEDA, 2024 |
| C4Plants | Proportion of land covered by C4 photosynthetic pathway plants | CEDA, 2024 |
| CSoil | Amount of carbon in the soil | CEDA, 2024 |
| CVeg | Amount of carbon in vegetation | CEDA, 2024 |
| FHarvest | Amount of carbon being passed into the atmosphere from crop harvesting | CEDA, 2024 |
| FLuc | Net carbon flux of carbon to the atmosphere resulting from land use changes | CEDA, 2024 |
| Lai | Leaf area index | CEDA, 2024 |
| NBP | Net biome production | CEDA, 2024 |
| NPP | Net primary production | CEDA, 2024 |
| RhLndSum | Total heterotrophic respiration on the land surface | CEDA, 2024 |
| ShrubFrac | Proportion of land covered by shrubs | CEDA, 2024 |
| TreeFrac | Proportion of land covered by trees | CEDA, 2024 |

Table 3 Input Parameter Descriptions

| Input_Parameter | Description | Reference |
|---|---|---|
| alpha | Quantum efficiency | McNeall et al, 2024 |
| a_wl | Allometric coefficient - relates woody biomass to lai | McNeall et al, 2024 |
| bio_hum_cn | Carbon to Nitrogen ratio microbial biomass and humified pools | McNeall et al, 2024 |
| b_wl | Allometric exponent - relates woody biomass to lai | McNeall et al, 2024 |
| dcatch_dlai | Rate of change - canopy capacity lai | McNeall et al, 2024 |
| dqcrit_io | Critical humidity deficit | McNeall et al, 2024 |
| dz0v_dh | Rate of Change - vegetation roughness | McNeall et al, 2024 |
| f0 | Maximum ratio - internal to atmospheric CO2 | McNeall et al, 2024 |
| fd | Scale factor - Dark respiration | McNeall et al, 2024 |
| g_area | Disturbance rate | McNeall et al, 2024 |
| g_root | Turnover rate - root biomass | McNeall et al, 2024 |
| g_wood | Turnover rate - woody biomass | McNeall et al, 2024 |
| gs_nvg | Surface conductance | McNeall et al, 2024 |
| hw_sw | Ratio - nitrogen stem to nitrogen heartwood - TRY database | McNeall et al, 2024 |
| kaps_roth | Soil respiration rate - RothC submodel | McNeall et al, 2024 |
| knl | Decay of nitrogen through canopy - function of lai | McNeall et al, 2024 |
| lai_max | Maximum lai | McNeall et al, 2024 |
| lai_min | Minimum lai | McNeall et al, 2024 |
| lma | Leaf mass per unit area | McNeall et al, 2024 |
| n_inorg_turnover | Lifetime of inorganic nitrogen pool | McNeall et al, 2024 |
| nmass | Top leaf nitrogen content per unit mass | McNeall et al, 2024 |
| nr | Root nitrogen concentration | McNeall et al, 2024 |
| retran_l | Fraction of retranslocated leaf nitrogen | McNeall et al, 2024 |
| retran_r | Fraction of retranslocated root nitrogen | McNeall et al, 2024 |
| r_grow | Growth respiration fraction | McNeall et al, 2024 |
| rootd_ft | Root depth | McNeall et al, 2024 |
| sigl | Specific density of leaf carbon | McNeall et al, 2024 |
| sorp | Leaching of inorganic nitrogen through soil profile | McNeall et al, 2024 |
| tleaf_of | Temperature bellow which leaves are dropped | McNeall et al, 2024 |
| tlow | Lower temperature for photosynthesis | McNeall et al, 2024 |
| tupp | Upper temperature for photosynthesis | McNeall et al, 2024 |
| l_vg_soil | Switch for van Genuchten soil hydraulic model | McNeall et al, 2024 |

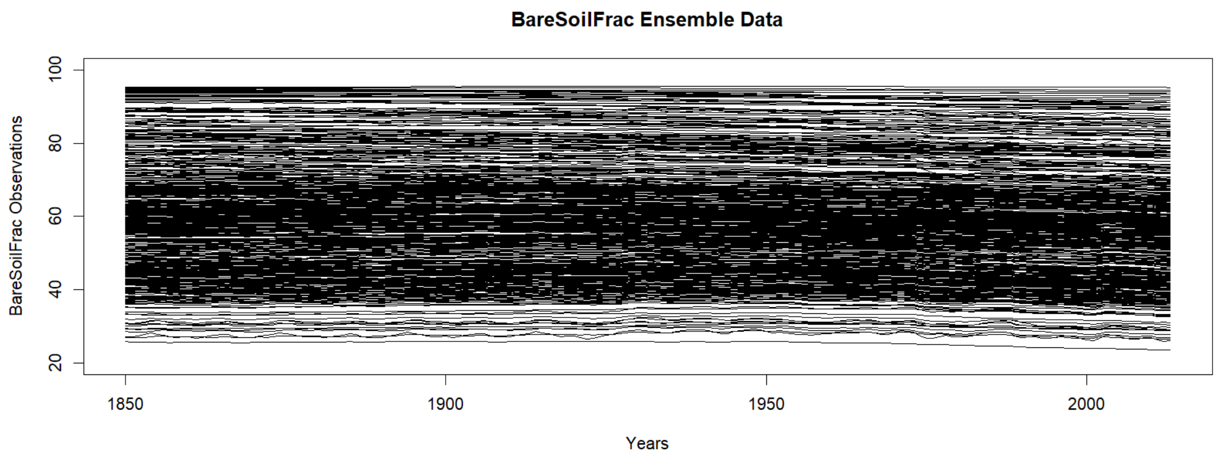

Figure 1: Ensemble Data Example BareSoilFrac

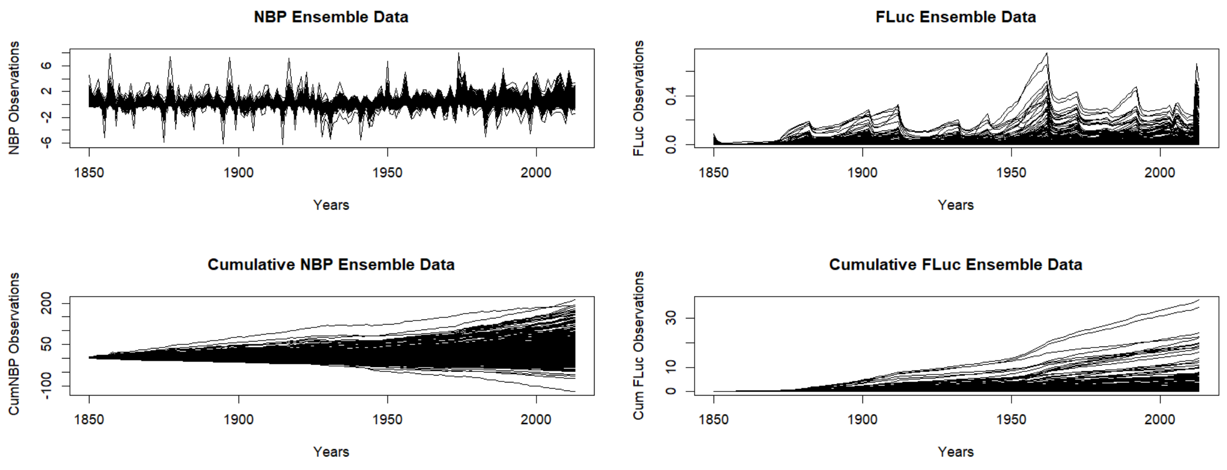

Figure 2: Before and after cumulative transformation of NBP and FLuc ensemble data

Therefore, initial analysis first involved plotting the data which also allowed for the discovery of non-linear patterns within NBP and FLuc. Where figure 1 represents an example of ensemble data in this case for BareSoilFrac after removal of zero carbon ensemble members. Whilst figure 2 represents the original and transformed versions of NBP and FLuc.

From visualising the data, it was suggested each ensemble member exhibits a relatively stable linear pattern within the BareSoilFrac dataset. Furthermore, the comparison between NBP and FLuc with their transformed version implied an improvement in ensemble linearity thereby providing evidence supporting the use of the cumulative versions for pca reduction.

To understand the process of Gaussian process emulator construction the first 20 years of the data was initially used to create simple models for a smaller time frame to reduce runtime. During this process different covariance functions were tested to determine suitability. This process used the out of box hyperparameter values the model selected allowing for just the effect of the covariance function to be determined. From the testing the Matern5/2 function had the best performance and to ensure it would be suitable overall the data used was extended to the full range with the same result being observed. Additionally, grid search and random search were initially trialled where the grid search performed marginally better than the out of box hyperparameter selection whilst the random search performed worse. However, further research led to the discovery of the Bayesian optimisation method for hyperparameter tuning and its eventual implementation in the full modelling process. This testing process allowed for initial exploration of the Gaussian process method on the ensemble data before pca reduction was used.

Following this process, the random forest and XGBoost methods were also initially explored though to a lesser extent. This involved determining which hyperparameters were the most important to tune where in the case of the random forest the mtry and ntree parameters were found to have the biggest effect whilst in the case of XGBoost if was found all six hyperparameters were of importance. Initial testing of the random forest utilised plotting of the mtry parameter to understand how error changed based on the value used. Typically, two patterns were found to occur where within the first pattern a l-shape was formed representing how error decreased largely at first but as the value of mtry increased the effect on error plateaued suggesting smaller values would be more suitable for the full analysis. Comparatively the second pattern exhibited was a U-shape which suggested in some instances if the value of mtry is too large this may result in overfitting resulting in increased error thereby also supporting the use of smaller values for the full analysis to prevent model overfitting.

Additionally, for the XGBoost initial exploration further research was required compared to the random forest due to its increase in complexity. The purpose of this was to ensure the correct data setup when being fed into the model alongside finding initial hyperparameter values to test. Where the XGBoost method is unable to utilise the typical data frame structure in R. However, this led to the discovery of using the caret packages grid search method as this effectively allowed me to bypass the data setup process for the XGBoost model and instead using a setup like that of the random forest thereby simplifying the process. This discovery then led to the caret grid search method implementation into the full analysis allowing both the hyperparameter tuning and model training process to be combined smoothly.

Furthermore, initial pca reduction was explored without using the cumulative transformations however, this led to 83 principal components being required to retain 99.9% of variance within NBP and 20 for FLuc. Whilst after cumulative transformation this was reduced to 5 each which substantially reduced the number of models needed particularly for NBP. Comparatively, it was found typically only 1 or 2 principal components were needed for other datasets where this likely stemmed back to figure 1 with most ensemble members having a stable structure where most of the variance can be captured with few linear combinations. This in turn allowed for substantial dimensionality reduction of the initial 164 observations per ensemble member leading to significant decreases in computational cost overall.

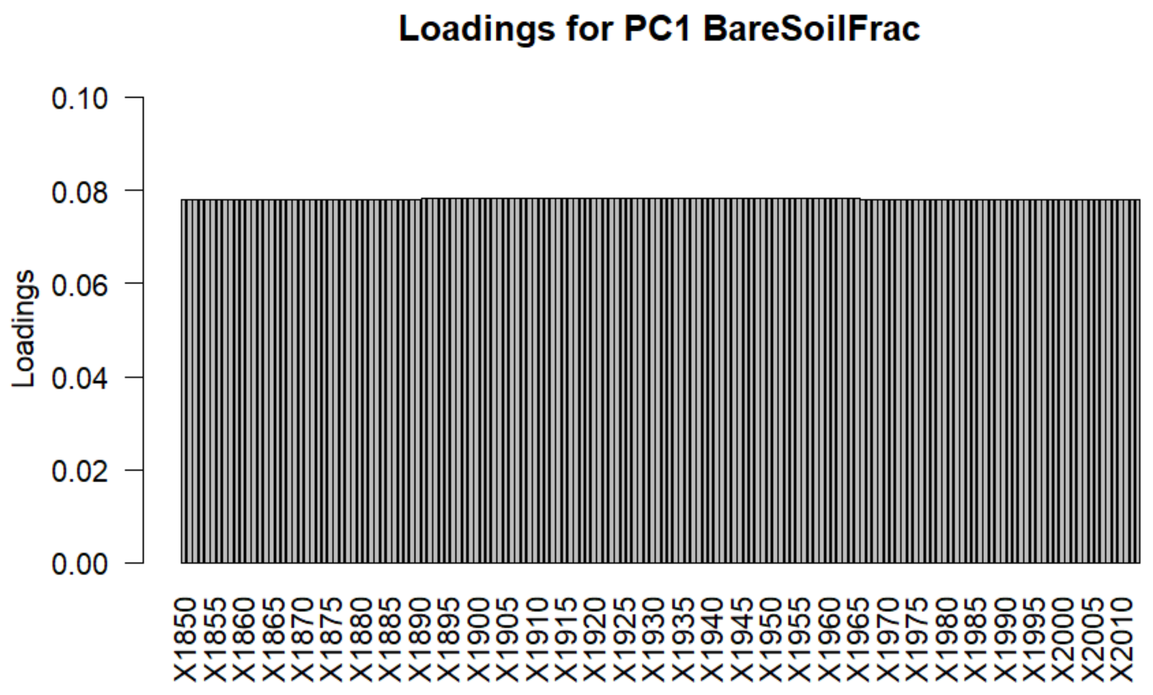

Figure 3: Principal component loadings for BareSoilFrac PC1 for each year

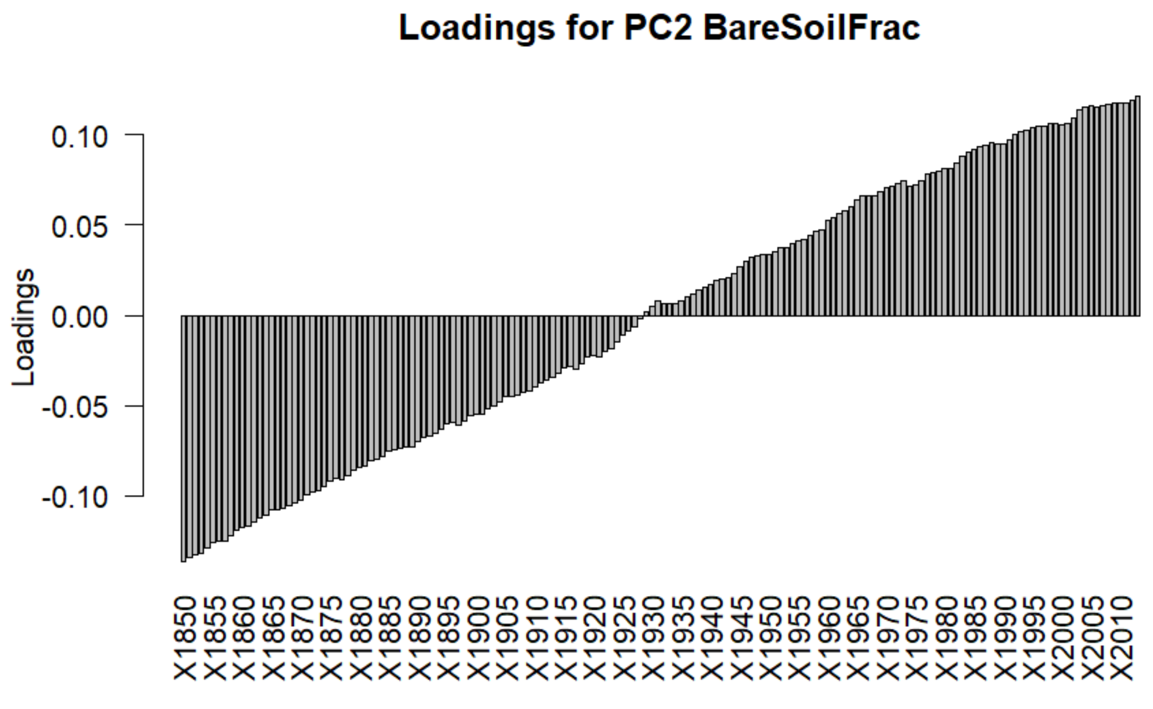

Figure 4: Principal component loadings for BareSoilFrac PC2 for each year

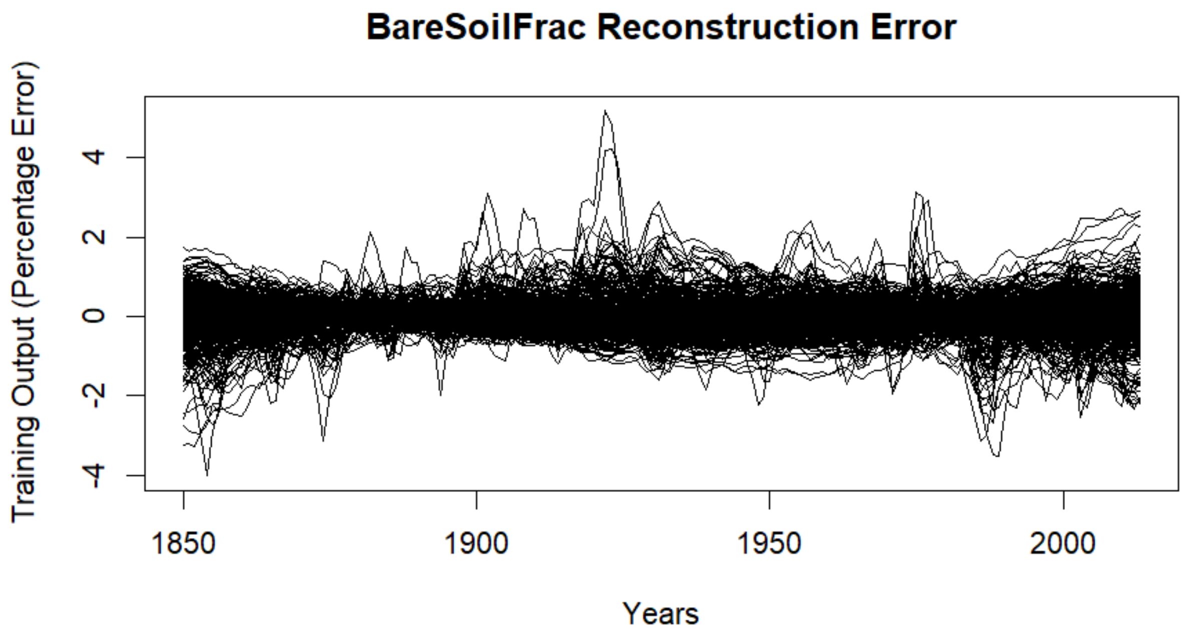

Figure 5: Associated information loss for the reconstruction of the BareSoilFrac dataset with fewer principal components

Additionally figure 3 represents the loading's of the first principal component for the BareSoilFrac dataset which represents a uniform distribution implying the first principal component may represent an evenly distributed global pattern, trend or average that does not favour a specific time period. Comparatively, figure 4 represents the loading's of the second principal component which implied a gradient pattern with a transition point around 1925. Where data in the earlier years implied an opposing effect compared to data in the later years where overall it appears to capture a long term trend within the data. Where these shifts are possibly due to land use changes as a result of the industrial revolution.

Furthermore, figure 5 represented the error associated with reconstructing the BareSoilFrac dataset into the original scaling (as seen in figure 1) using fewer principal components. The error maximums were 4.5 and -4% though this applied to only a few ensemble members compared to the total 399 within the training split with most ensembles deviating between 2% and -2% of their original values implying just 2 principal components were capable of capturing most of the variance.

Following this initial exploration the information discovered was collated allowing for the overall method to be determined. This led to a smooth process of tuning and training models and producing interpretable results for analysis.

## Results

### Ensemble Mean Plots

As stated previously three different methods have been used for producing emulators for the JULES land surface model with these being: Gaussian Processes, random forests and XGBoost. Using these three methods emulators were produced for each of my thirteen datasets with a total of 84 emulators being produced with 28 for each method.

The ensembles plots produced used the mean for each ensembles the purpose of this was for improved clarity where the summary of these plots implied the Gaussian process and XGBoost methods produced similar overall performance with the XGBoost method performing slightly better overall whilst the random forest method performed the worst. However, as the mean represented the general performance of the ensembles it doesn't capture the behaviour of each member individually therefore following this, plots within the pca space were produced to represent the individual behaviour of data points.

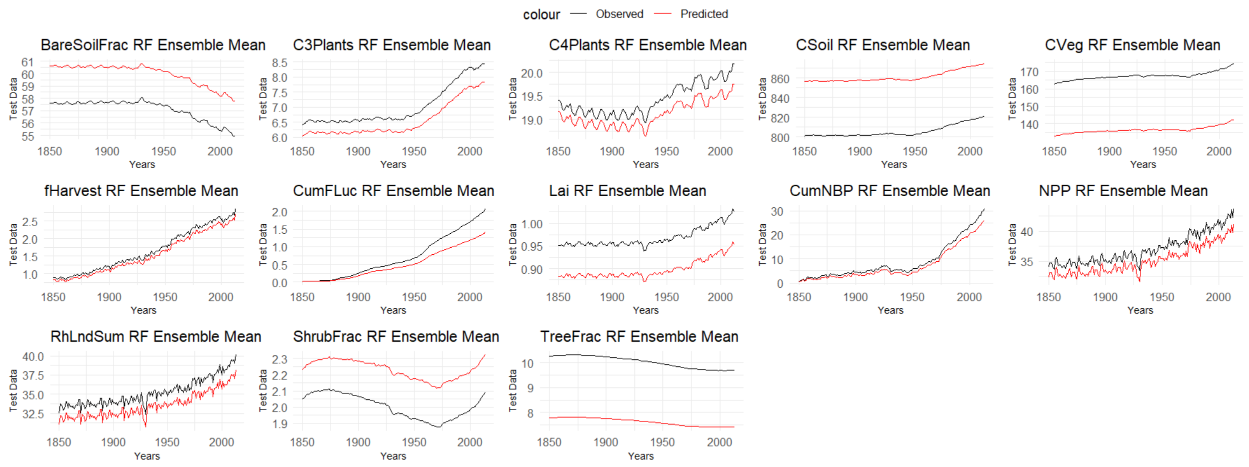

Figure 6: randomForest ensemble mean observed vs predicted plot within original dimensions

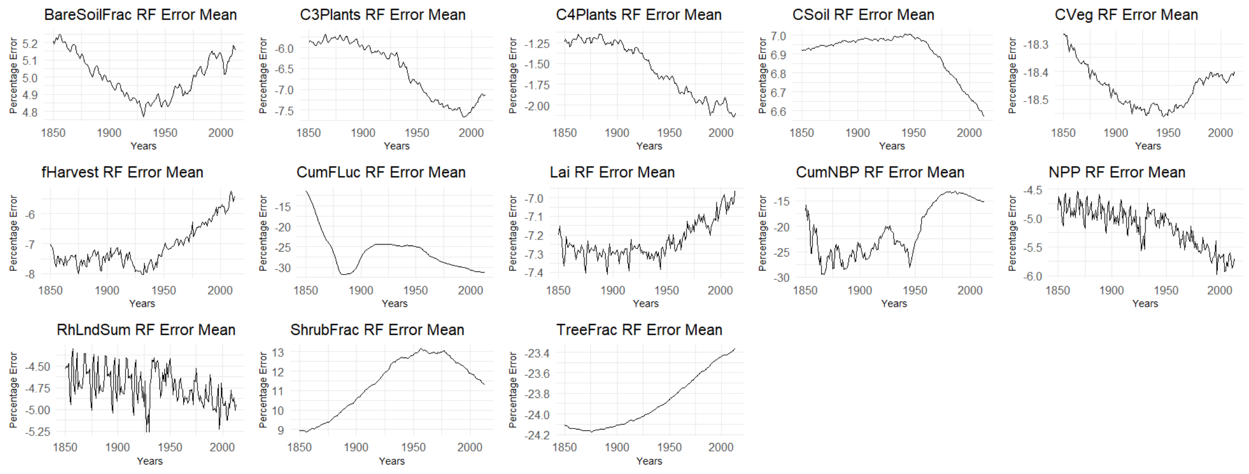

Figure 7: randomForest ensemble mean percentage error plot

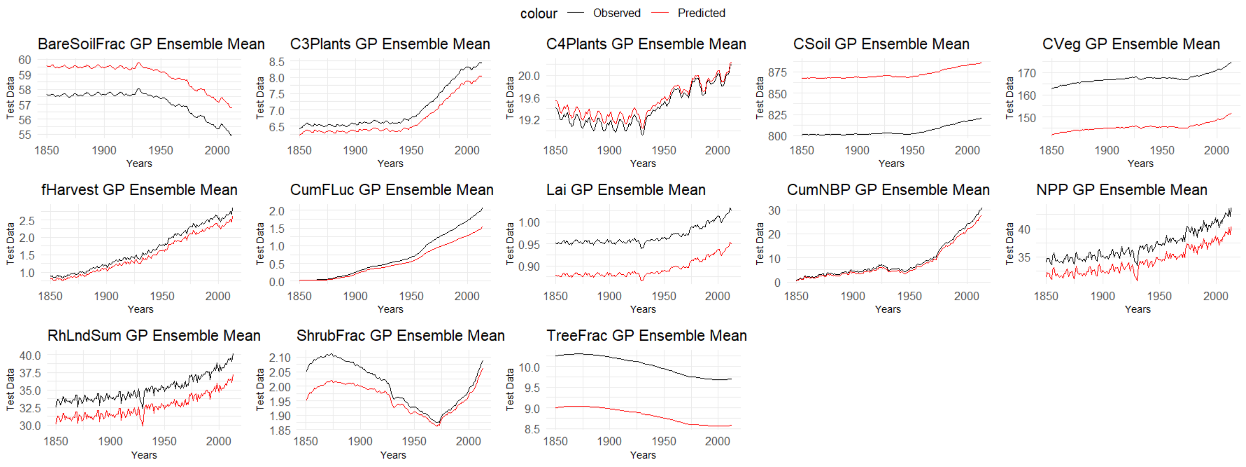

Figure 8: Gaussian Process ensemble mean observed vs predicted plot within original dimensions

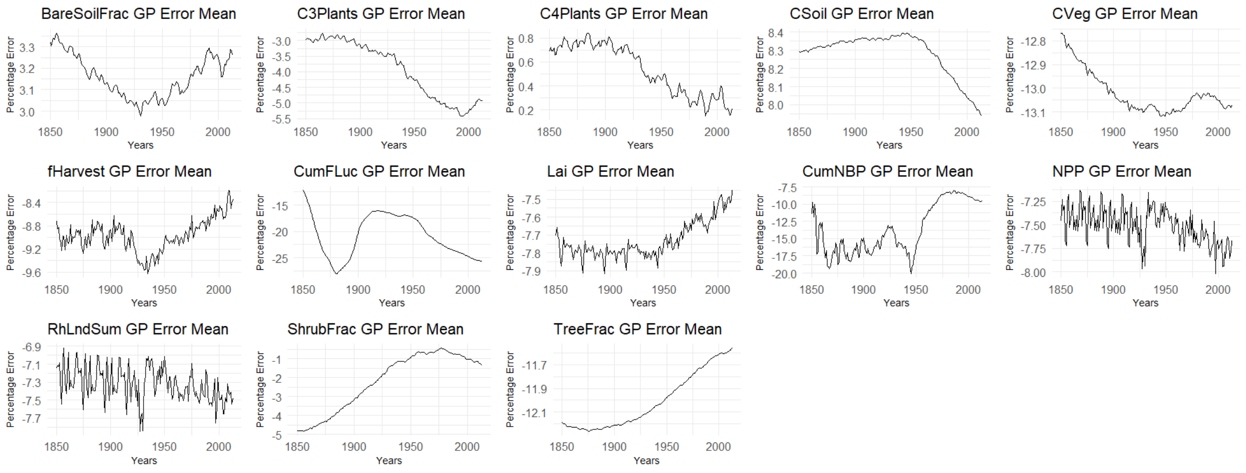

Figure 9: Gaussian Process ensemble mean percentage error plot

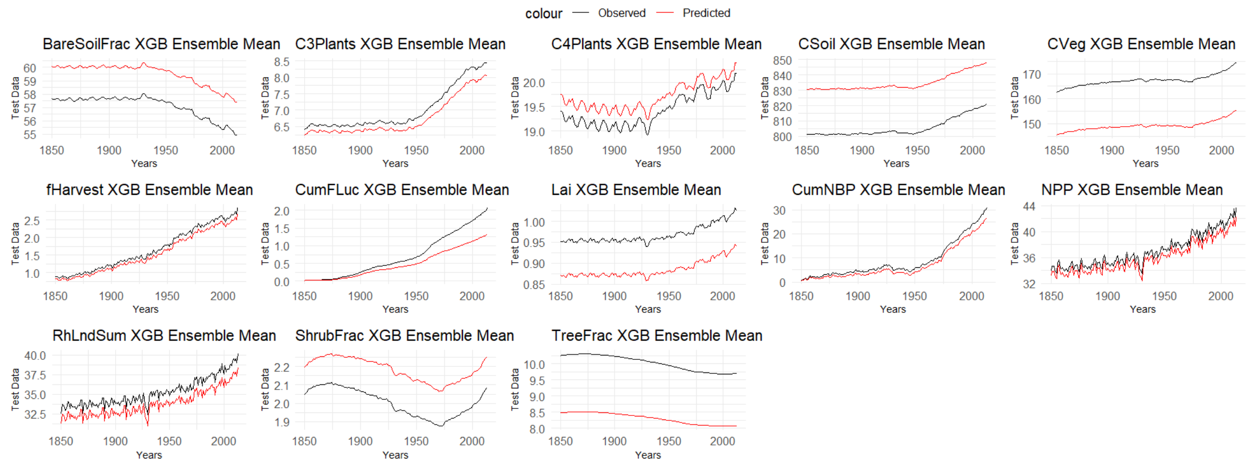

Figure 10: Extreme Gradient Boosting Machine ensemble mean observed vs predcited plot within original dimensions

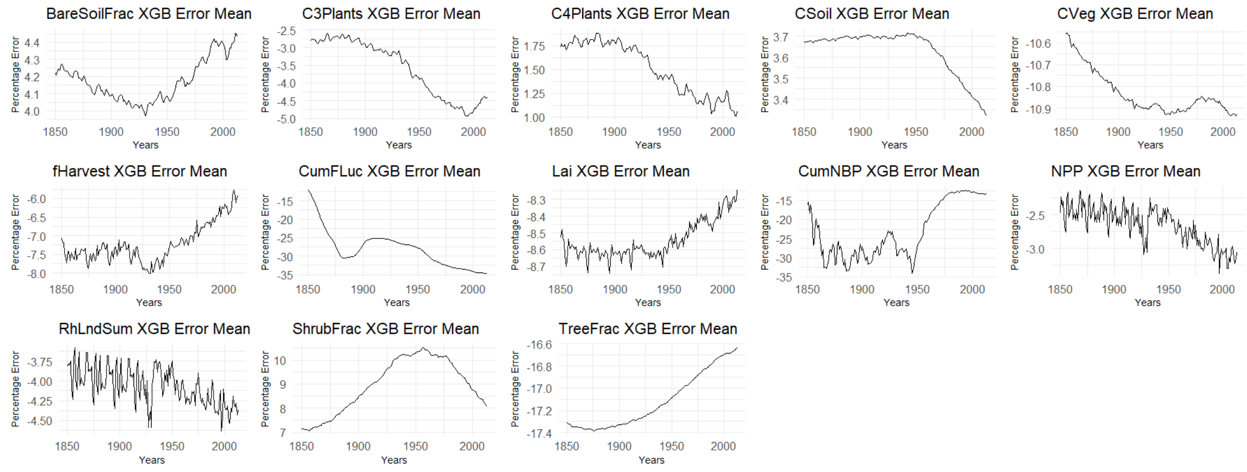

Figure 11: Extreme Gradient Boosting Machine ensemble mean percentage error plot

Initially, plots were produced to visualise how well the predictions compare to the true values under the original space with the plots representing the ensemble mean to understand how the models were performing on average. Figures 6 and 7 represent the random forest predictions and associated errors with figures 8 and 9 representing the Gaussian process predictions and associated errors while figures 10 and 11 represent the XGBoost predictions and associated errors. Where the errors have been represented as percentages to allow comparison between both datasets and modelling method.

For figure 6 it was implied the random forest, on average, tended to under predict more so than over predict where there were only three instances of implied over predicting which were for the BareSoilFrac, CSoil and ShrubFrac datasets. Based on these plots it was implied the FHarvest and CumNBP datasets seemed to have the most accurate predictions however, to better understand accuracy figure 7 was produced. Which instead implied the C4Plants dataset had the most accurate predictions on average as the error ranged from -1.1% to -2.25%. This suggested on average the random forest model was most accurate for this dataset. Whilst the other datasets had the following ranges: BareSoilFrac 4.75% to 5.25%, C3Plants -5.75% to -8%, CSoil 6.55% to 7%, CVeg -18.25% to -18.55%, FHarvest -5.5% to -8%, CumFLuc -12.5% to -32.5%, Lai -6.95% to -7.4%, CumNBP -12.5% to -30%, NPP -4.5% to -6%, RhLndSum -4% to -5.25%, ShrubFrac 9% to 13% and TreeFrac -23.4% to -24.2% where it was suggested CumFLuc had the lowest overall accuracy closely followed by CumNBP. Additionally, the overall error patterns implied the most error is exhibited at the start and end of the ensembles although some plots suggested error was highest around the middle of the ensembles. From this it was suggested C4Plants, BareSoilFrac and RhLndSum had on average high accuracy due to average errors being 5% or lower whilst C3Plants, CSoil, FHarvest, Lai and NPP had moderate accuracy due to average error being between 5%-10% with all other datasets exhibiting low accuracy due to average errors being above 10%.

Similarly to figure 6 figure 8 implied the Gaussian process, on average, also typically underestimated its predictions with only BareSoilFrac, C4Plants and CSoil being over predicted. Additionally, it was also implied visually C4Plants, FHarvest and CumNBP had the most accurate predictions. Where figure 9 supports this implication for C4Plants with the average error being between 0.2% and 0.8% however, this implication is proven incorrect for FHarvest and CumNBP which had error ranges of -8.2% to -9.6% and -7.5% to -20% respectively. With all other datasets having ranges of: BareSoilFrac 3.35% to 3%, C3Plants -2.75% to -5.5%, CSoil 7.95% to 8.4%, CVeg -12.75% to -13.1%, CumFLuc -12.5% to -25%, Lai -7.45% to -7.9%, NPP, -7.125% to -8%, RhLndSum -6.9% to -7.8%, ShrubFrac -0.5% to 5% and TreeFrac -11.6% to -12.2%. Alike the random forest the error patterns also exhibit a pattern where generally the highest error is observed at the start or end of the ensembles with some datasets having error peak around the middle of the ensembles. Furthermore, C4Plants, BareSoilFrac, C3Plants and ShrubFrac had errors of 5% or less suggesting high accuracy, CSoil, FHarvest, Lai, NPP and RhLndSum having between 5%-10% errors suggesting moderate accuracy whilst all other datasets had low accuracy due to errors greater than 10%. Furthermore, the Gaussian process is suggested to be more accurate than the randomForest as all datasets except CSoil, FHarvest, Lai, NPP and RhLndSum had on average lower errors than the Gaussian process. Suggesting overall the Gaussian process performed better on greater than 50% of the datasets compared to the random forest.

Similarly to the previous methods figure 10 suggested the XGBoost also tended to underestimate as opposed to overestimate with only BareSoilFrac, C4Plants, CSoil and ShrubFrac being overestimated on average. Additionally, fHarvest, CumNBP and NPP are visually suggested to have the most accurate predictions with this implication only being implied for NPP by figure 11 Due to an error range of -2.25% to -3.25% implying high accuracy whilst this implication is implied to be incorrect for FHarvest and CumNBP with error ranges of -5.75% to -8% implying moderate accuracy and -12.5% to -35% implying low accuracy respectively. The ranges for the other datasets were as follows: BareSoilFrac 4% to 4.45% implying high accuracy, C3Plants -2.5% to -5% implying high accuracy, C4Plants 1% to 1.975% implying high accuracy, CSoil 3.35% to 3.7% implying high accuracy, CVeg -10.55% to -10.95% implying low accuracy, CumFLuc -12.5% to -35% implying low accuracy, Lai -8.25% to -8.75% implying moderate accuracy, RhLndSum -3.625% to -4.625% implying high accuracy, ShrubFrac 7% to 10.5% implying moderate accuracy and TreeFrac -16.6% to -17.4% implying low accuracy. The same error patterns were also exhibited with most data points at the start and end having the highest error alongside some data points in the middle of the ensembles. Additionally, for BareSoilFrac the model performed better than the random forest but worse than the Gaussian process, for C3Plants it performed the best, for C4Plants it performed worse than the Gaussian process but better than the random forest, for CSoil it performed the best, for CVeg it performed the best, for FHarvest it performed better than the Gaussian process but the same as the randomForest, for CumFLuc it performed the worst, for Lai it performed the worst, for CumNBP it performed the worst, for NPP it performed the best, for RhLndSum it performed the best, for ShrubFrac it performed better than the random forest but worse than the Gaussian process and for TreeFrac it performed better than the random forest but worse than the Gaussian process.

Overall, the random forest performed the best on 2 but the worst on 6 datasets the Gaussian process performed best on 6 but worst on 4 datasets whilst the XGBoost performed best on 6 but worst on 3 datasets. With the XGBoost and random forest both being the best for FHarvest. It is implied overall the random forest performed the worst whereas the performance between the Gaussian process and XGBoost being similar although the Gaussian process performed worse on one additional dataset compared to the XGBoost.

## Predictions Within PCA Space

To summarise the pca plots overall, they suggested that predictions were typically not as accurate as depicted by some of the mean ensemble plots which is highlighted by the typically large RMSE values associated with each method. Where in general the Gaussian process and XGBoost methods performed evenly with the random forest producing the worst predictions. It was also suggested out of all datasets CSoil had the best performance.

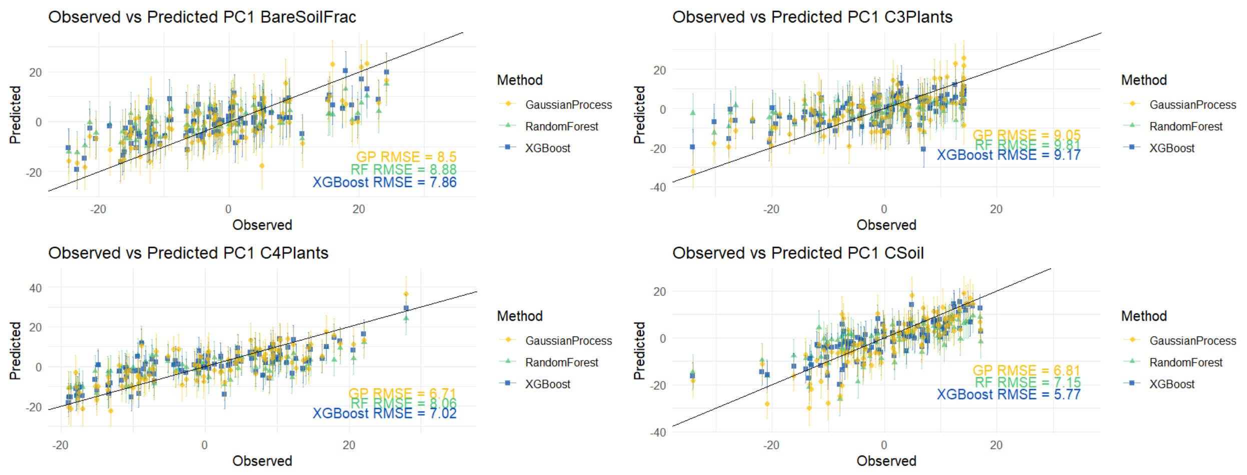

Figure 12: Observed vs Predicted plot within principal component space for all methods

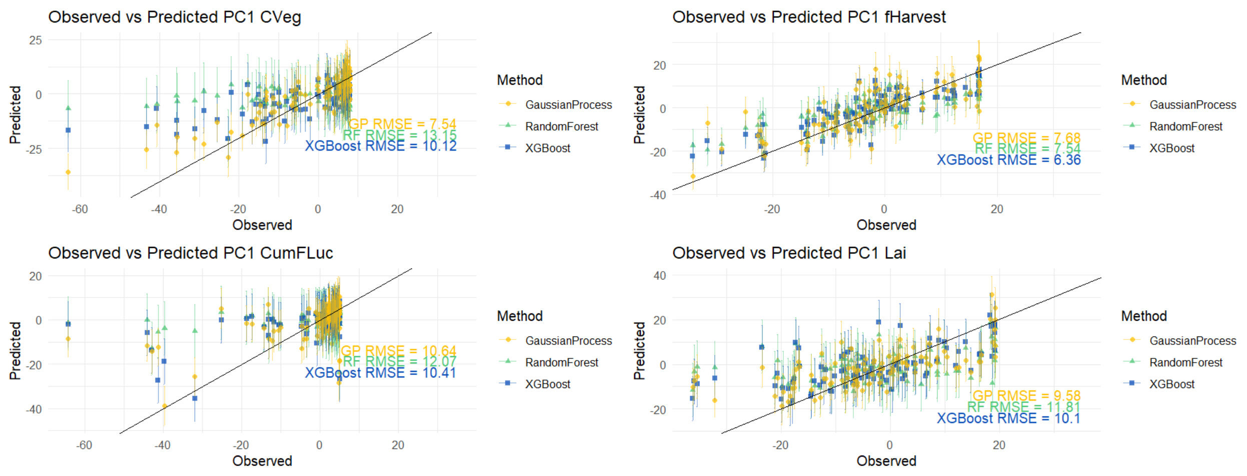

Figure 13: Observed vs Predicted plot within principal component space for all methods

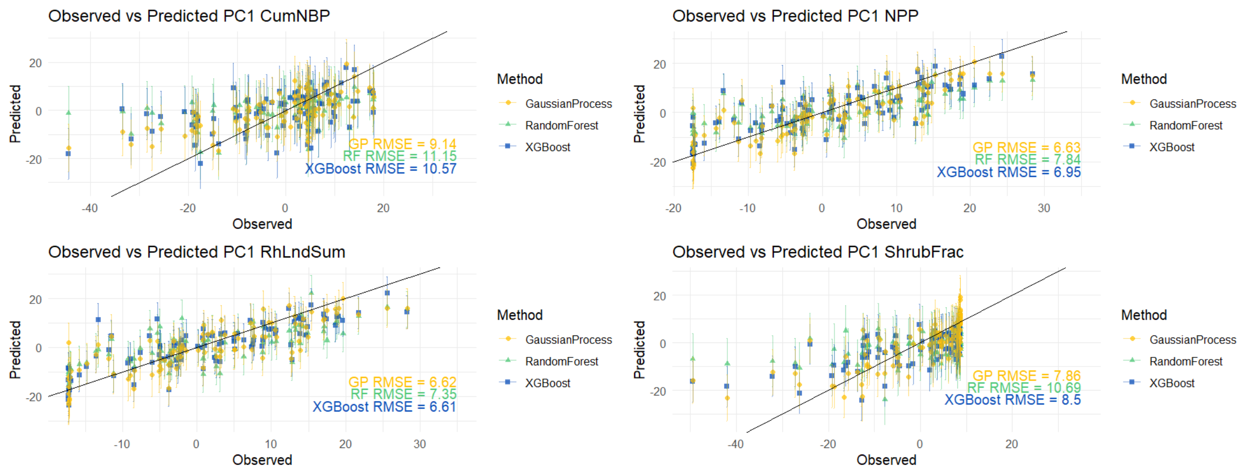

Figure 14: Observed vs Predicted plot within principal component space for all methods

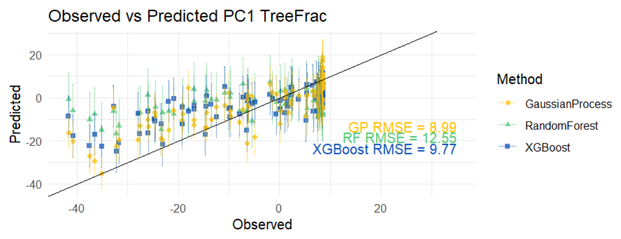

Figure 15: Observed vs Predicted plot within principal component space for all methods

Figures 12, 13, 14 and 15 provided error metrics for the predictions based on the reduced space created by principal component analysis where only the first principal component is investigated as it holds most of the variance. These plots have been produced so it can be understood how the model predictions are performing whilst in the reduced space produced by principal component analysis.

For figure 12 BareSoilFrac, it is implied within the reduced space overall over predictions are implied to be more frequent due to most of the model points lying above the reference line. Additionally, the model RMSEs are as follows: Gaussian Process 8.50, random forest 8.88 and XGBoost 7.86. Implying that the predictions within the reduced space were better captured on average compared to the other methods. Although has implied by the plot the predictions themselves are still rather poor. Comparatively, for C3Plants the predictions are implied to be more balanced between over and under predicting with the RMSE values as follows: Gaussian process 9.05, random forest 9.81 and XGBoost 9.17 implying the Gaussian process performed slightly better on average compared to the other methods. Similarly, the predictions for C4Plants also appear to be mostly balanced within the reduced space between under and over predicting with the RMSEs as follows: Gaussian process 6.71, random forest 8.06 and XGBoost 7.02. Implying the Gaussian process again performed slightly better than the other two methods. Finally, the predictions for CSoil are also implied to be reasonably balanced with the RMSE values as follows: Gaussian process 6.81, random forest 7.15 and XGBoost 5.77. Implying the XGBoost performed slightly better than the other two methods. It is also implied on average the models performed slightly better for CSoil due to the overall lower RMSE values compared to the other datasets in this grid.

Whereas for figure 13, CVeg there is also an implied balance between under and over predicting within the reduced space although it is implied the models struggled with this dataset due to the vertical conglomeration of points between x0 and x10. Furthermore, the RMSE values are as follows: Gaussian process 7.54, random forest 13.15 and XGBoost 10.12 where this implied the Gaussian process was better at making predictions on average for this dataset compared to the other methods. Comparatively for FHarvest, the models had less issues with making predictions for this dataset although there is an implied minor bias towards over predicting due to more points overall being above the reference line. Additionally, the RMSE values are as follows: Gaussian process 7.68, random forest 7.54 and XGBoost 6.36 which implies the XGBoost was slightly better on average with its predictions than the other methods. Similarly to CVeg, the models also struggled with the CumFluc dataset due to a vertical conglomeration of points around x0 and x10 with it being unclear as to proportion of predictions being over or under predicted. However, the RMSE values are as follows: Gaussian process 10.64, random forest 12.07 and XGBoost 10.41 suggesting it performed marginally better than the other methods particularly the Gaussian process. Furthermore, for the Lai dataset the models had fewer issues with making predictions with there being a suggested balance between under and over predicting. Where the RMSE values are as follows: Gaussian process 9.58, random forest 11.81 and XGBoost 10.1 where it is implied the Gaussian process performed marginally better than the other methods particularly regarding the XGBoost. It is also implied that for this grid the models performed best on the FHarvest dataset due to the models having their lowest RMSE values for this dataset.

Furthermore, for figure 14 the CumNBP dataset appears to imply a reasonably balanced distribution of over and under predicted points although there are some implications of issues at x-20 and x5 due to some vertical point conglomerations occurring. However, the RMSE values were as follows: Gaussian process 9.14, random forest 11.15 and XGBoost 10.57, implying the Gaussian process performed on average slightly better for this dataset. Similarly, the NPP dataset suggests reasonable balance between over and under predicting points. Where the RMSE values are as follows: Gaussian process 6.63, random forest 7.84 and XGBoost 6.95 implying the Gaussian process performed marginally better than the other two methods. Comparatively, for the Rh_Land_Sum dataset there is a suggested minor bias towards under predicting due to more points falling bellow the reference line. Where the RMSE values are as follows: Gaussian process 6.62, random forest 7.35 and XGBoost 6.61 where this implied that both the Gaussian process and XGBoost performed to a similar level. However, for the SHrubFrac dataset there are also implied model struggles with a conglomeration of vertical points being between x0 and x10 where it is also suggested the models had a bias towards under predicting for this dataset due to most points being bellow the reference line. With the RMSE values being as follows: Gaussian process 7.86, random forest 10.69 and XGBoost 8.5 which implied the Gaussian process performed slightly better for this dataset than the other methods. It is also implied the models performed on average slightly better for the Rh_Land_Sum dataset due to it having the lowest overall RMSE values.

Finally, figure 15 for the TreeFrac dataset also implies reasonable balance between under and over predicting however it is implied the models struggled with this data set due to the conglomeration of vertical points between x0 and x10. However, the RMSE values were as follows: Gaussian process 8.99, random forest 12.55 and XGBoost 9.77 implying the Gaussian process performed slightly better than the other methods for this dataset. Overall, between all datasets it is suggested the models performed best for the CSoil dataset within the reduced space due to it having the lowest overall RMSE values for each model. With the models also performing well on the Rh_Land_Sum and NPP datasets which shared similar RMSE values.

## Rank Histograms

Furthermore, rank histograms, figures 16, 17 and 18, have been used to understand where the models are making their predictions based on the range of possible values. The ranks have also been normalized to allow for comparison between datasets and not just between models.

Overall, the rank histograms highlighted some instances of uniformity however typically biases were present where the random forest typically had middle bias suggesting conservative predictions while the Gaussian process and XGBoost models implied more left and right skewing implying over and underprediction biases respectively where these biases were partly more pronounced for the Gaussian process.

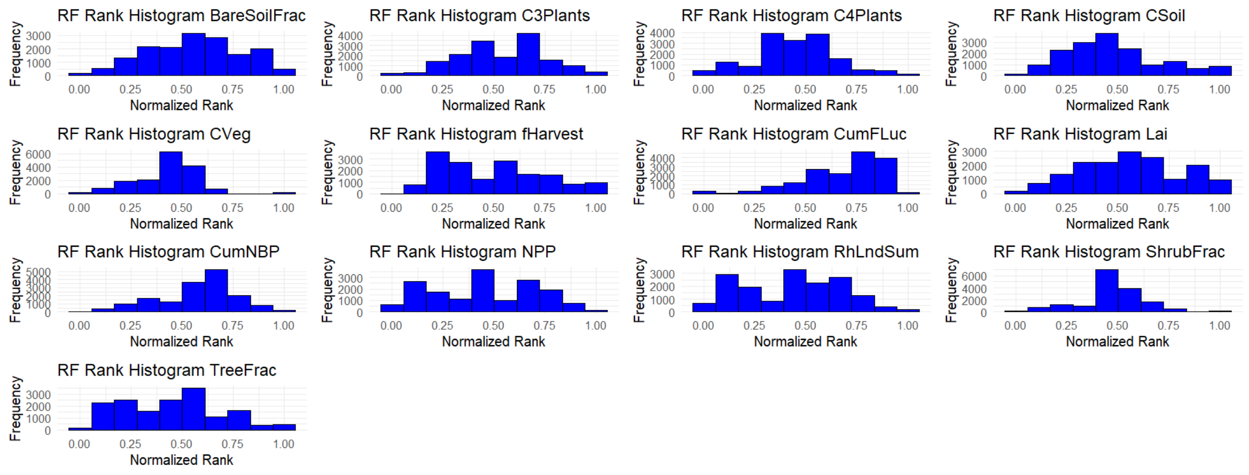

Figure 16: randomForest normalized rank histograms

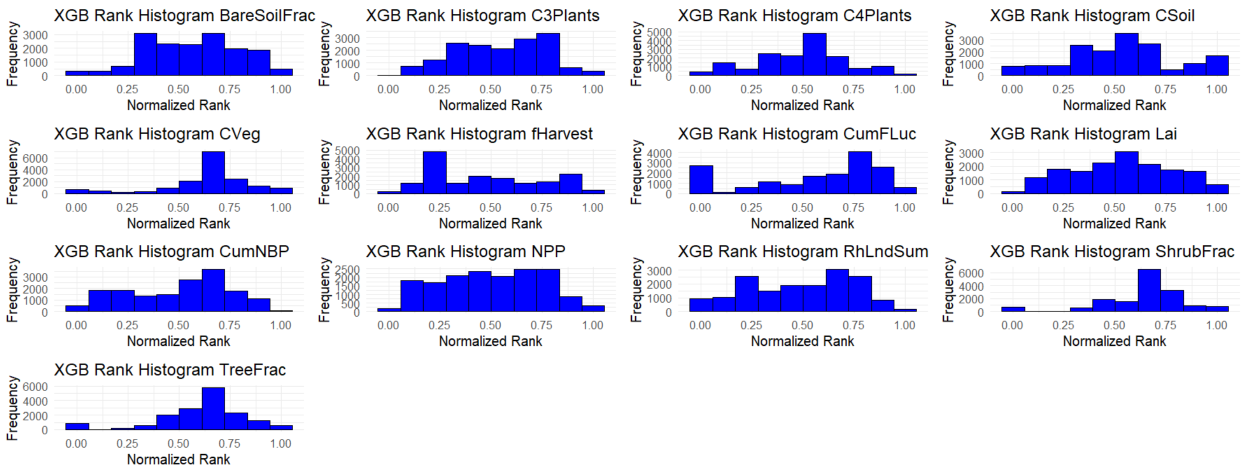

Figure 17: Extreme Gradient Boosting Machine normalized rank histograms

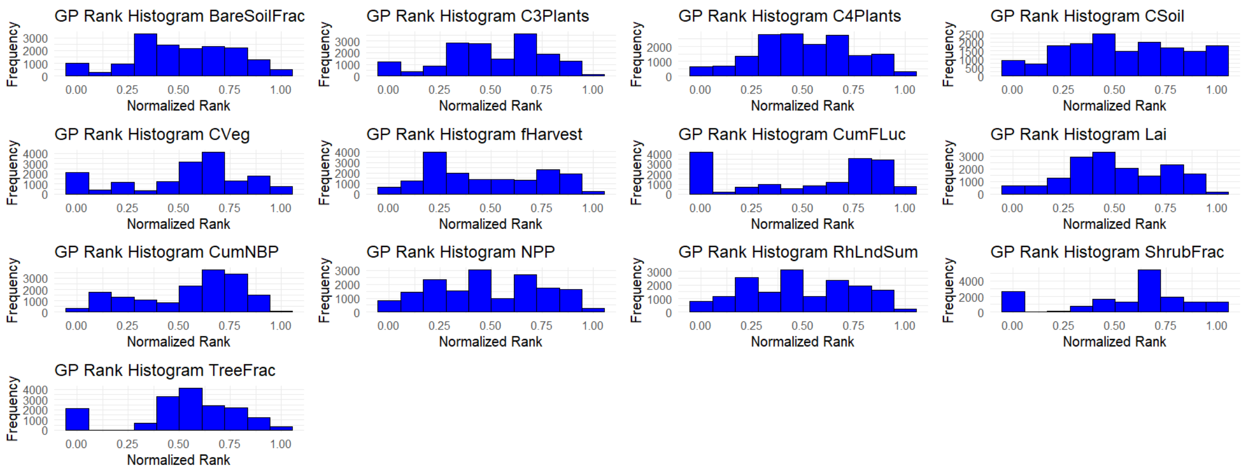

Figure 18: Gaussian Process normalized rank histograms

Thereby figure 16 the random forest rank histograms suggested for most dataset's predictions are being mostly made around the mean/median of the possible true values this is seen in the following datasets: BareSoilFrac, C3Plants, C4Plants, CSoil, CVeg, FHarvest, Lai, NPP, Rh_Land_Sum, ShrubFrac and TreeFrac. This might be due to random forests having an averaging effect when making predictions or else it may be a result of the distribution of the data with more ensembles centered around the mean. Comparatively, the CumFLuc and CumNBP histograms seem to suggest a more right skewed distribution which is likely a result of using the cumulative distribution which results in more larger observations resulting in the model being more likely to then pick higher values from its range of possible values.

Similarly to the random forest figure 17 the XGBoostrank histograms also mostly exhibit a middle skew this is mostly seen in the following datasets: BareSoilFrac, C3Plants, C4Plants, CSoil, Lai, CumNBP, NPP and Rh_Land_Sum where this is also likely due to most ensemble observations being close together however as XGBoost typically has a regularization effect and depending on its strength this may also skew predictions towards the middle. Additionally, some of the datasets exhibit right skewing these being: CVeg, CumFLuc, ShrubFrac and TreeFrac which may be a result of the additive boosting process XGBoost undergos when making predictions, unlike the averaging effect of the random forest. Furthermore, FHarvest seems to potentially exhibit a somewhat left skew due to a single peak around 0.25. Additionally, it should also be noted unlike the random forest several predictions have often been made at the lowest end of possible values where this is particularly noticeable for CumFLuc.

Furthermore, figure 18 the Gaussian process rank histograms also exhibit mostly middle skews with this being the case for the BareSoilFrac, C3Plants, C4Plants, CSoil, Lai, NPP, Rh_Land_Sum and TreeFrac datasets. Whereas CVeg, CumFLuc, CumNBP and ShrubFrac exhibit a more right skewed distribution and FHarvest having a slightly left skewed distribution. Also, alike the XGBoost there are many predictions being made at the lower end of possible values with this being more prevalent for the Gaussian process compared to XGBoost.

## Measure of Uncertainty

Additionally, to understand the uncertainty behind each models' predictions the mean confidence interval widths were calculated and represented as percentages. The results of this are outlined in table 4 with the lowest value being highlighted in green. Where the uncertainty can be used as a measure of precision to understand how precise a model is. Where table 5 outlines how well the model's prediction intervals capture the true values from the testing dataset expressed as a percentage.

Overall, the uncertainty values implied the random forest had the lowest levels of uncertainty with the Gaussian process and XGBoost performing similarly with the XGBoost being slightly less uncertain. Where the Lai dataset produced the lowest uncertainties and Cumulative NBP the greatest uncertainties. Comparatively, all models were implied to not truly capture the true values of the testing datasets but the Gaussian process model is implied to better capture the true values of the test datasets where it had a maximum coverage of 36% however, the maximum coverage's were higher for both the random forest and XGBoost methods 42.2% and 40.4% where all the maximum coverages were for the CumNBP dataset.

Table 4 Percentage uncertainty (CI Widths)

| Mean_Width_GP | Mean_Width_RF | Mean_Width_XGB | Dataset |
|---:|---:|---:|---|
| 5.37 | 5.22 | 5.15 | BareSoilFrac |
| 3.50 | 3.35 | 3.43 | C3Plants |
| 2.79 | 2.26 | 2.47 | C4Plants |
| 34.09 | 32.25 | 32.09 | CSoil |
| 8.85 | 8.52 | 8.91 | CVeg |
| 3.16 | 3.21 | 3.20 | fHarvest |
| 2.31 | 1.64 | 1.75 | CumFLuc |
| 0.11 | 0.11 | 0.11 | Lai |
| 44.90 | 35.84 | 42.37 | CumNBP |
| 15.89 | 15.52 | 16.14 | NPP |
| 11.54 | 11.46 | 11.55 | Rh_land_sum |
| 0.46 | 0.38 | 0.42 | ShrubFrac |
| 1.15 | 0.82 | 0.97 | TreeFrac |

Table 5 True Testing Data Value Coverage Percentage by Model and Dataset

| Coverage_Percentage_GP | Coverage_Percentage_RF | Coverage_Percentage_XGB | Dataset |
|---:|---:|---:|---|
| 8 | 0.0 | 0.0 | BareSoilFrac |
| 10 | 2.0 | 0.5 | C3Plants |
| 4 | 0.0 | 0.0 | C4Plants |
| 1 | 0.0 | 0.0 | CSoil |
| 3 | 0.0 | 2.0 | CVeg |
| 26 | 4.0 | 3.5 | fHarvest |
| 36 | 14.8 | 22.6 | CumFLuc |
| 12 | 0.0 | 0.0 | Lai |
| 36 | 42.2 | 40.4 | CumNBP |
| 16 | 0.0 | 0.0 | NPP |
| 13 | 0.0 | 1.5 | Rh_land_sum |
| 4 | 0.0 | 1.5 | ShrubFrac |
| 2 | 4.0 | 0.0 | TreeFrac |

Where for BareSoilFrac the XGBoost method had the lowest uncertainty 5.15%. Although the XGBoost had the lowest overall al three models performed equally with only very minor differences in uncertainty.

Similarly, for C3Plants all three uncertainties are close together with the random forest being the lowest at 3.35%. This pattern is also implied with C4Plants with again the uncertainties being close together with the random forest having the lowest 2.26%.

Comparatively, there is a slightly bigger difference between uncertainties for CSoil where the XGBoost had the lowest uncertainty 32.09% which is similar for the random forest model however, the Gaussian process had a noticeable worse uncertainty of 34.09%. It can also be noted overall uncertainties were much higher for this dataset suggesting the model precision is much worse for CSoil compared to the previous datasets. Furthermore, the uncertainty is suggested to be lower for CVeg along with being more consistent across models where the random forest had the lowest uncertainty 8.52%.

Additionally, for FHarvest the uncertainties are consistent across models with the Gaussian process having the lowest uncertainty 3.16%. Similarly, CumFluc exhibits the same pattern as with CSoil where the Gaussian process was found to be slightly noticeably higher than the other methods where the random forest had the lowest uncertainty 1.64%.

Furthermore, for Lai all uncertainties are equal suggesting very similar precision for each method with this dataset also having the overall lowest uncertainty between datasets with 0.11%.

Comparatively, CumNBP had the largest uncertainties between dataset sets but noticeably the random forest exhibited the lowest uncertainty 35.84% compared to the other methods with over 40% uncertainty. Whereas performance for NPP was partly better with the random forest also having the lowest uncertainty 15.52%. Additionally, Rh_Land_Sum also exhibits consistency across model uncertainties again with the random forest having the lowest 11.46%.

Furthermore, for SHrubFrac uncertainties are also relatively consistent across models also with the random forest having the lowest 0.38%. Where TreeFrac also implied relatively consistent uncertainties again with the random forest having the lowest 0.82%.

It was therefore implied the random forest had the overall lowest uncertainties however all model uncertainties are relatively similar with the only exception being CumNBP where the random forest performed notably better than the other methods.

Additionally, the overall ability of the models to capture the true values of the testing set within the confidence interval ranges were very poor with maximum coverages of 36% for the Gaussian process, 42.2% for the random forest and 40.4% for XGBoost.

However, the Gaussian process was implied to better capture the true values across all datasets having the most coverage for 11 datasets followed by the random forest with the most coverage for 2 datasets whilst XGBoost did not have the best coverage for any dataset.

Furthermore, the random forest and XGBoost methods were found to occasionally have no true values fall between the 95% confidence interval of their predictions where this occurred for BareSoilFrac, C4Plants, CSoil, Lai, and NPP for both methods with the random forest additionally having no true values within the range for RhLndSUm and ShrubFrac datasets and XGBoost no true values within the range for the TreeFrac dataset also. Comparatively the Gaussian process had at minimum 1% coverage.

This implied that the confidence interval ranges were too narrow overall where although the Gaussian process was found to have the largest ranges this proved to be more useful as its overall coverage's were generally better than the other methods.

## Computational Cost

However, with accuracy and precision metrics having been discussed computational cost is also another important factor in the emulation process as the goal of emulators is not just to accurately produce predictions but to also do so at a greater rate than the original simulation. Therefore, to compare computational costs a clustered bar graph has been produced to not only compare between methods but also between datasets.

In general, the random forest was found to be the fastest method to tune and train followed by the Gaussian process with the XGBoost being the slowest. Additionally, in some instances the Gaussian process run times were comparable and occasionally faster than the random forest.

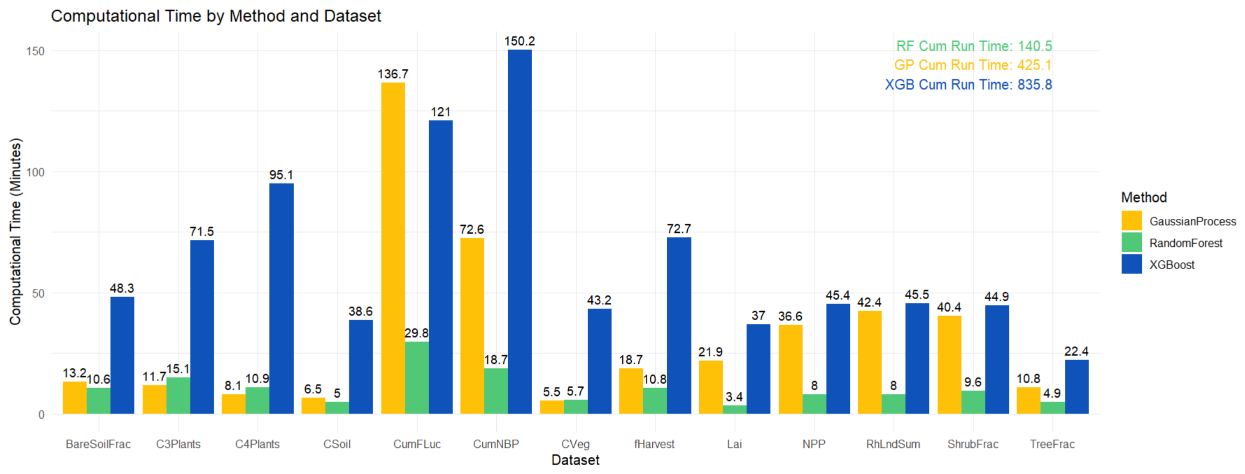

Figure 19: Clustered bar plot of computational time alongside values for cumulative run time for each method

Therefore, figure 19 represents the culmination of computational cost represented as the time taken to tune and train the models. Overall, the random forest model implies the lowest computational cost with run times ranging from 3.4 to 29.8 minutes with the Gaussian process having similar run times for BareSoilFrac, C3Plants, C4Plants, CSoil and CVeg but being noticeably longer for the other datasets with run times ranging from 5.5 to 136.7 minutes. Furthermore, XGBoost the noticeably longest run times for all datasets except for CumFluc, where the Gaussian process had the longest run time, with run time ranging from 22.4 to 150.2 minutes. With the mean and cumulative run times for each data set being 10.8 and 140.5 minutes for the random forest, the Gaussian process being 32.7 and 425.1 minutes and the XGBoost being 64.3 and 835.8 minutes. Where deviations in computational time between datasets is a result of the differing number of principal components used between datasets.

Where it can be implied that the difference in run times between datasets is likely due to the number of principal components where both CumFLuc and CumNBP had the most principal components of 5 with all other datasets having between 1 and 2 principal components. Where the number of principal components correlates to the number of models needed. Whereas the difference between run times between models may stem from the number of hyperparameters tuned during the tuning process, where the XGBoost had the most hyperparameters of 6, alongside the overall complexity behind each method.

## Limitations and Sensitivities

However, there were associated limitations/sensitivities with the process, namely the methods used for hyper parameter tuning. Where the random forest underwent tuning for one hyperparameter and the Gaussian process underwent tuning for two hyperparameters whereas the XGBoost underwent tuning for six hyperparameters. Where this would notably cause discrepancies in computational time however, it still highlights that when choosing a model, the number of hyperparameters to tune may influence model choice based on associated increases in computational time. The reason for the difference in hyperparameters counts is not just to the number of parameters available but due to whether the algorithms used supported the tuning of certain hyperparameters where for example the use of the caret R package for tuning the random forest only allows tuning of the “mtry” parameter where to tune other parameters a custom function that caret supports would need to be engineered however this is a complex process which is why it was not undertaken.

Additionally, the methods used to tune each process were different resulting in further discrepancy with a grid search being performed for the random forest and gradient boosting machine and Bayesian optimization for the Gaussian process. Where this inconsistency may have led to differences in relative tuning times due to each process having differing levels of associated computational cost. Where grid search requires more evaluations compared to Bayesian optimization which may have led to comparably longer tuning processes. Particularly because Bayesian optimization also handles greater dimensions more efficiently as grid search cost increases exponentially as the dataset dimensions increase.

Furthermore, there were limitations when calculating the error bars for the predictions made within the principal component space (figures 12, 13, 14 and 15). This is due to the Gaussian process model containing the associated standard deviations whereas the random forest and XGBoost did not have this functionality. Although the standard deviation can be calculated for these two methods this is not possible for models with only one principal component. Therefore, for these two methods the error was taken form the residuals resulting with only a single error margin for all predictions as opposed to the unique error margins for each prediction for the Gaussian process.

Comparatively, there were also limitations when representing the ensemble member predictions as a focus on clarity was used as opposed to the full behavioural representation. In this regard as the mean was used this only provides information on general performance but fails to highlight the model's predictive capabilities for extreme values. Thereby, the full predictive capabilities of the models were not fully represented. However, the implementation of full ensemble plots would be difficult in terms of managing a suitable resolution whilst also allowing the data to be represented clearly thus highlighting the difficulties in representing large volumes of data.

## Summary, Discussion and Future Steps

### Summary

Overall, the results suggest no clear answer with no one method having performed the best within all the categories therefore, all performance metrics will need to be considered individually.

Such as the random forest having had the highest overall mean error with its ensemble predictions whilst comparatively the Gaussian process and XGBoost boasted similar performance. Therefore, the best performing method for this category was XGBoost due to having the lowest performance on the fewest number of datasets. Although this can only be considered as an assumption as the lack of a full ensemble visualisation prevents a fully informed decision.

Furthermore, the pca space observations also suggested similar behaviours with the Gaussian process and XGBoost performing similarly whilst the random forest performed the worst. Overall, due to the Gaussian process having the lowest RMSEs for 8 datasets compared to the Gaussian processes 5 with this metric the Gaussian process was implied to have the best performance.

However, the rank histograms do not imply a clear best method due to each method exhibiting some type of bias this typically being middle skewing for the random forest and left/right skewing for the Gaussian process and XGBoost methods. Because of this no clear method can be differentiated as being the best using this metric.

Comparatively, the uncertainty measures did suggest a clearer best method which was implied to be the random forest which had the lowest uncertainties for 10 datasets whilst XGBoost and the Gaussian process had the lowest uncertainty for 3 and 2 datasets respectively. However, although the random forest was implied to have the lowest uncertainty as the accuracy of the emulators is implied to be typically poor it is difficult to justify a method being the best in terms of precision if the overall predictions themselves were poor.

Although, despite the smaller confidence interval ranges the random and XGBoost methods did not capture the true values as well as the Gaussian process model. This implied that despite having larger intervals if the larger intervals capture more true values then it is more useful than having smaller interval ranges that do not capture the true values suggesting the Gaussian process performed better in this regard. Although it can still be noted that the maximum coverage value acquired was still higher for both the random forest and XGBoost methods compared to the Gaussian process.

Furthermore, the computational times also show a clear differentiation between methods with the random forest being the fastest, followed by the Gaussian process then the XGBoost being the slowest. Where this metric implies in terms of computational cost the random forest is the least costly to tune and train however, alike the uncertainty if the overall accuracy of predictions is low then a faster computation time holds little value if the method is found to be less suitable.

Therefore, by combining the outcomes of these different metrics it can be considered the best method is between the Gaussian process and the XGBoost. Thereby, the best method may depend on time availability where, my suggestion is if time is less constrained the XGBoost may contain the most potential however, if time is constrained the Gaussian process would likely be more suitable. But overall, no clear method shows adequate ability to emulate JULES based on the data used.

However, the results of this project link back to McNeall et al, 2024s study where the data used for this project was specifically designed to be exploratory with the purpose of reducing the input parameter space. This project successfully highlights this requirement and provides evidence for why this is needed where improvements in the simulation will likely produce data more adequate for emulation purposes. Where this requirement was implied by the overall poor performance of all methods used suggesting the problem of emulating JULES may not necessarily lie with the methods themselves but the data currently available for emulator training.

Additionally, these results tie into the other studies previously discussed. Where studies producing models for satellite data typically acquired better results such as Yang et al, 2017s study where their random forest model produced an r-squared of 0.967 whilst comparatively studies which focussed on emulation such as Owen and Liuzzo, 2019 where NS coefficients of 0.64 and 0.70 were acquired which indicated satisfactory and good performance respectively. This suggested that data quality and availability are key aspects of good model performance where satellite based data is typically plentifully due to the amount of data they are able to acquire within shorter time frames meaning poor data can be omitted more readily whilst comparatively due to the slow nature of simulators especially those which are complex such as with JULES data availability is lower meaning poor outputs cannot be readily discarded.

Thereby, the results of this project and the study this project builds upon both highlight the need for improvements in simulations due to the process of producing emulators being a topic of current interest within the realms of environmental sciences.

## Future Steps

However, as with all projects there are aspects to which can be improved upon thereby, I shall highlight areas for improvements and further steps which could be considered in the future. Where the first aspect I would improve is the tuning process of the models specifically for the random forest and XGBoost methods wherein instead of using grid search I would instead replace this with Bayesian optimisation. The purpose of this is due to the computational cost of performing a grid search for a large dataset being costly whereas Bayesian optimisation would have been more efficient more the dataset size I was using. Additionally, this would have allowed for a fairer comparison between computational times rather than the possible differentiations being due to the tuning method used the differentiation would have fully been between the different methods. Also, as done in Salter et al, 2019s study, applying singular value decomposition could be considered for the initial dimensionality reduction of the outputs.

Furthermore, as implied by this project the XGBoost and Gaussian processes suggested the most potential thereby, to further expand upon these alterations differing versions of both could be used. These could be achieved through a multivariate approach where each output observations were represented as years this would likely result in a strong correlation between output features suggesting the use of a multivariate approach could be suitable however, a notable change in computational cost would be applied as a multivariate approach without pca would likely result in longer tuning and training times. Additionally, the Gaussian process can further be elaborated upon where the standard version used for this project is typically not suitable for large datasets due to the complexity in the inversion of the covariance matrix. Therefore, different versions could be more applicable where examples for future testing could incorporate sparse or local Gaussian processes where in essence both these types revolve around sub-setting the data resulting in reduced computational times.

Additionally, another area for improvement would be to find a method that represents the individual ensemble members more effectively, rather than solely using the overall mean which was done for this project. One approach could involve dividing the ensemble members into subsets based on their values. For example, if there are 100 ensemble members, you could create 10 regions, each containing 10 members where you could calculate the mean of the members in each of the 10 regions. This would result in 10 different means, providing more detailed information than a single overall mean while maintaining clarity. Where additionally, exploring methods to visualize all ensemble members individually, while still preserving clarity and resolution, could offer further improvements. This would allow for a more comprehensive representation of the data, capturing the full range of variability within the overall ensemble.

As a final note, an addition to this project could aim to further expand upon the input dimensions to isolate features with the most importance using sensitivity analysis. Where as outlined in Baker et al, 2022s study Gaussian processes can provide preliminary information to determine which of the input parameters have the greatest importance. This could also further improve upon the computational cost by reducing the number of computations needed by reducing the number of input parameters used. Additionally, this could help improve performance for the ensemble machine learning methods in this case random forest and XGBoost where fewer parameters to test would also improve upon computational time. Also in the case of the random forest it may improve prediction accuracy this is due to the tuning parameter mtry which randomly selects combinations of input features to use in the decisions made meaning the method may end up being more probable to select key features.

## References

Baker E, Harper A.B, Williamson D and Challenor P, 2022. Emulation of high-resolution land surface models using sparse Gaussian processes with application to JULES. Geoscientific Model Development, 15(5), pp.1913-1929.

CEDA (Centre for Environmental Data Analysis) 2024, CMIP6 Data request version 01.00.33, Available at: https://cmip6dr.ceda.ac.uk/vocab/var/

Chen T and Guestrin C, 2016. Xgboost: A scalable tree boosting system. In Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining (pp.785-794).

Chen T, He T, Benesty M, Khotilovich V, Tang Y, Cho H, Chen K, Mitchell R, Cano I, Zhou T, Li M, Xie J, Lin M, Geng Y, Li Y and Yuan J, 2024. Xgboost: Extreme Gradient Boosting. R package version 1.7.7.1, https://CRAN.R-project.org/package=xgboost

Chu C, Bartlett M, Wang Y, He F, Weiner J, Chave J and Sack L, 2016. Does climate directly influence NPP globally? Global Change Biology, 22(1), pp.12-24.

Clark D.B, Mercado L.M, Sitch S, Jones C.D, Gedney N, Best M.J, Pryor M, Rooney G.G, Essery R.L.H, Blyth E, Boucher O, Harding R.J, Huntingford C and Cox P.M: The Joint UK Land Environment Simulator (JULES), model description – part 2: Carbon fluxes and vegetation dynamics. Geoscientific Model Development, 4, 701-722, https://doi.org/10.5194/gmd-4-701-2011, 2011.

Fisher R.A and Koven C.D, 2020. Perspectives on the future of land surface models and the challenges of representing complex terrestrial systems. Journal of Advances in Modelling Earth Systems, 12(4), p.e2018MS001453.

Golding N and Purse B.V, 2016. Fast and flexible Bayesian species distribution modelling using Gaussian processes. Methods in Ecology and Evolution, 7(5), pp.598-608.

Heimann M and Reichstein M, 2008. Terrestrial ecosystem carbon dynamics and climate feedbacks. Nature, 451(7176), pp.289-292.

Higdon D, Gattiker J, Williams B and Rightley M, 2008. Computer model calibration using high-dimensional output. Journal of the American Statistical Association, 103(482), pp.570-583.

Jia Y, Jin S, Chen H, Yan Q, Savi P, Jin Y and yuan Y, 2021. Temporal-spatial soil moisture estimation from CYGNSS using machine learning regression with a preclassification approach. IEEE Journal of Selected Topics in Applied Earth Observation and Remote Sensing, 14, pp.4879-4893.

Keetz L.T, Aalstad K, Fisher R.A, Teran C.P, Naz B.S, Pirk N, Yilmaz Y and Skarpaas O, 2024. Inferring parameters in a complex land surface model by combining data assimilation and machine learning. Authorea Preprints. Liaw A and Wiener M, 2002. Classification and Regression by randomForest. R News 2(3), 18-22.

McNeall D, Robertson E and Wiltshire A, 2024. Constraining the carbon cycle in JULES-ES-1.0. Geoscientific Model Development, 17(3), pp.1059-1089.

Meiyappan P and Jain A.K, 2012. Three distinct global estimates of historical land-cover change and land-use conversions for over 200 years. Frontiers of Earth Science, 6, pp.122-139.

Owen N.E and Liuzzo L, 2019. Impacts of land use on water resources via a Gaussian process emulator with dimension reduction. Journal of hydroinformatic, 21(3), pp.411-426.

Pal S and Sharma P, 2021. A review of machine learning applications in land surface modelling. Earth, 2(1), pp.174-190.

Pan S, Pan N, Tian H, Friedlingstein P, Sitch S, Shi H, Arora V.K, Haverd V, Jain A.K, Kato E and Lienert S, 2020. Evaluation of global terrestrial evapotranspiration by state of the art approaches in remote sensing, machine learning and land surface models. Hydrology and Earth System Sciences, 24(3), pp.1485-1509.

R Core Team (2024). R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

Roustant O, Ginsbourger D and Deville Y, 2012. DiceKriging, DiceOptim: Two R Packages for the Analysis of Computer Experiments by Kriging-Based Metamodeling and Optimization. Journal of Statistical Software, 51(1), 1-55. https://www.jstatsoft.org/v51/i01/

Roxburgh S.H, Berry S.L, Buckley T.N, Barnes B and Roderick M.L, 2005. What is NPP? Inconsistent accounting of respiratory fluxes in the definition of net primary production. Functional Ecology, 19(3), pp.378.382.

Salter J.M and Williamson D, 2016. A comparison of statistical emulation methodologies for multi-wave calibration of environmental models. Environmetrics, 27(8), pp.507-523.

Salter J.M, Williamson D.B, Scinocca J and Kharin V, 2019. Uncertainty quantification for computer models with spatial outputs using calibration-optimal bases. Journal of the American Statistical Association.

## Appendix/Code

RF

```
library(tidyverse)
library(ggplot2)
library(reshape2)
library(randomForest)
library(ggpubr)
library(caret)
library(gridExtra)
library(beepr)
library(Metrics)
library(reshape2)
library(patchwork)

load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/Loop_data.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/functions.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/All_rf_lists.RData")

load("C:\\Users\\tjsla\\OneDrive\\Desktop\\Masters\\Dissertation\\ensemble-jules-historical-timeseries.RData")

#Remove unsuitable ensembles and calculate cumulative FLuc and NBP
nbp_ens_wave00[288, "1937"] <- nbp_ens_wave00[288, "1936"]
npp_ens_wave00[288, "1937"] <- npp_ens_wave00[288, "1936"]
cFLuc <-  t(apply(fLuc_lnd_sum_ens_wave00, 1, FUN = cumsum))
cnbp <-  t(apply(nbp_ens_wave00, 1, FUN = cumsum))

y_datasets<- list(y1=baresoilFrac_lnd_mean_ens_wave00,y2=c3PftFrac_lnd_mean_ens_wave00,
                  y3=c4PftFrac_lnd_mean_ens_wave00,y4=cSoil_ens_wave00,y5=cVeg_ens_wave00,
                  y6=fHarvest_lnd_sum_ens_wave00,y7=cFLuc,y8=lai_lnd_mean_ens_wave00,
                  y9=cnbp,y10=npp_ens_wave00,y11=rh_lnd_sum_ens_wave00,
                  y12=shrubFrac_lnd_mean_ens_wave00,y13=treeFrac_lnd_mean_ens_wave00)

#pca values were obtained by using "Summary()" on pca object
#- lists variance for each component alongisde cumulative varaince
#- find where cumVar is 99.9%
npcs_list<- c(2,2,2,1,1,2,5,1,5,2,2,2,1)

X_Train_List<- list()
X_Test_List<- list()
Y_Train_List<- list()
Y_Test_List<- list()

#Loop to shuffle datasets
for (i in seq_along(y_datasets)) {
  comp_data <- data.frame(X, y_datasets[[i]])

  set.seed(123)
  shuffled_data <- comp_data[sample(nrow(comp_data)), ]

  train_indices <- 1:399
  test_indices <- 400:499
  X_Train_List[[i]] <- shuffled_data[train_indices, 1:ncol(X)]
  X_Test_List[[i]] <- shuffled_data[test_indices, 1:ncol(X)]
  Y_Train_List[[i]] <- shuffled_data[train_indices, (ncol(X) + 1):ncol(shuffled_data)]
  Y_Test_List[[i]] <- shuffled_data[test_indices, (ncol(X) + 1):ncol(shuffled_data)]

}

pca_list<- list()
YTR_pca_list<- list()
YTS_pca_list<- list()
#Apply principal component analysis to all datasets
for (i in seq_along(Y_Train_List)) {
  npcs<- npcs_list[[i]]
  pca_list[[i]]<- prcomp(Y_Train_List[[i]],center = T,scale. = T)

  YTR_pca_list[[i]]<- as.data.frame(pca_list[[i]]$x[, 1:npcs])
  YTS_pca_list[[i]]<- as.data.frame(predict(pca_list[[i]],
                             newdata=Y_Test_List[[i]])[, 1:npcs])
}

rf_grids_list <- list()
time_taken_rf_list <- list()
grid <- expand.grid(mtry = c(1:32))
control <- trainControl(method = "cv", number = 5)

#Grid search setup for random forest
for (i in seq_along(YTR_pca_list)) {
  YTR_pca <- YTR_pca_list[[i]]
  X_Train <- X_Train_List[[i]]

  rf_grids <- list()

  time_taken_rf <- system.time({
    set.seed(123)
    for (j in 1:ncol(YTR_pca)) {
      rf_grids[[j]] <- train(x = X_Train, y = YTR_pca[, j], method = "rf",
                             trControl = control, tuneGrid = grid, metric = "RMSE")
    }
  })

  rf_grids_list[[i]] <- rf_grids
  time_taken_rf_list[[i]] <- time_taken_rf

  print(paste("Dataset", i, "processed"))
}
beep(3)

plot(rf_grids_list[[13]][[1]])

mtry_list<- c(8,8,8,8,4,10,6,8,5,10,10,5,8)
ntree_list<- c(19,19,15,17,13,14,20,11,11,19,15,20,14)

rf_models_list<- list()
```

```r
#Train RF models based on tuned parameters
for (i in seq_along(YTR_pca_list)) {
  YTR_pca <- YTR_pca_list[[i]]
  X_Train <- X_Train_List[[i]]

  models <- list()
  set.seed(123)
  for (j in 1:ncol(YTR_pca)) {
    models[[j]] <- randomForest(x = X_Train, y = YTR_pca[, j],
                               mtry = mtry_list[i], ntree = ntree_list[i])
  }

  rf_models_list[[i]] <- models
}
plot(rf_models_list[[1]][[1]])

all_predictions_rf <- list()
all_actual_rf <- list()
#Acquire predictions for each model for each dataset
for (i in seq_along(rf_models_list)) {
  X_Test <- X_Test_List[[i]]
  Y_Test <- YTS_pca_list[[i]]
  rf_models <- rf_models_list[[i]]

  predictions_rf <- matrix(NA, nrow = nrow(X_Test), ncol = length(rf_models))
  for (j in 1:length(rf_models)) {
    rf_model <- rf_models[[j]]
    predictions_rf[, j] <- predict(rf_model, newdata = X_Test)
  }

  all_predictions_rf[[i]] <- predictions_rf
  all_actual_rf[[i]] <- Y_Test
}

rf_plot_data_list <- list()
#setup rf predictions for plotting
for (i in seq_along(all_predictions_rf)) {
  predictions <- all_predictions_rf[[i]]
  actual <- all_actual_rf[[i]]

  j <- 1
  if (ncol(predictions) >= j) {
    rf_predicted <- predictions[, j]
    rf_actual <- actual[, j]

    residuals <- rf_actual - rf_predicted
    std_error <- rep(sd(residuals), length(rf_predicted))

    rf_plot_data <- data.frame(
      Observed = as.numeric(rf_actual),
      Predicted = as.numeric(rf_predicted),
      Std_Error = std_error,
      Dataset = paste("Dataset", i),
      Component = paste("Component", j),
      Method = c("RandomForest")
    )

    rf_plot_data_list[[length(rf_plot_data_list) + 1]] <- rf_plot_data
  }
}

rf_rmse_list<- list()
#acquire pca space RMSEs for RF
for (i in seq_along(rf_models_list)) {
  X_Test <- X_Test_List[[i]]
  Y_Test <- YTS_pca_list[[i]]
  rf_models <- rf_models_list[[i]]

  predictions_rf <- predict(rf_models[[1]], newdata = X_Test)

  actual_rf <- Y_Test[, 1]
  rmse_value_rf <- rmse(actual_rf, predictions_rf)

  rf_rmse_list[[i]] <- rmse_value_rf

  print(paste("RMSE for Dataset", i, "Component 1:", rmse_value_rf))
}

mean_width_rf_list <- list()

#Function to calculate confidence interval mean widths and coverage percentage
compute_prediction_intervals <- function(X_Test, Y_Test, models, pca_obj) {
  predictions <- matrix(NA, nrow = nrow(X_Test), ncol = length(models))
  for (j in 1:length(models)) {
    model <- models[[j]]
    predictions[, j] <- predict(model, newdata = X_Test)
  }

  predicted_Y_Test <- predictions %*% t(pca_obj$rotation[, 1:ncol(predictions)])
  predicted_Y_Test <- scale(predicted_Y_Test, center = FALSE, scale = 1 / pca_obj$scale)
  predicted_Y_Test <- scale(predicted_Y_Test, center = -pca_obj$center, scale = FALSE)

  lower_quantile_rf <- apply(predicted_Y_Test, 1, quantile, probs = 0.05)
  upper_quantile_rf <- apply(predicted_Y_Test, 1, quantile, probs = 0.95)
  pred_intervals_matrix_rf <- data.frame(Lower_Quantile = lower_quantile_rf,
                                         Upper_Quantile = upper_quantile_rf)

  # Calculate interval widths
  interval_widths_rf <- pred_intervals_matrix_rf$Upper_Quantile - pred_intervals_matrix_rf$Lower_Quantile
  y_range <- max(Y_Test) - min(Y_Test)
  normalized_interval_widths_rf <- interval_widths_rf / y_range
  mean_width_rf <- mean(normalized_interval_widths_rf)
  mean_width_rf_percentage <- mean_width_rf * 100

  # Calculate the coverage percentage
  within_interval <- Y_Test >= pred_intervals_matrix_rf$Lower_Quantile & Y_Test
  <= pred_intervals_matrix_rf$Upper_Quantile
  coverage_percentage <- mean(within_interval) * 100

  return(list(mean_width_rf_percentage = mean_width_rf_percentage,
              coverage_percentage = coverage_percentage))
}

mean_width_rf_list <- list()
coverage_percentage_list <- list()

#acquire meanWidths and coverage widths for all datasets
for (i in seq_along(rf_models_list)) {
  X_Test <- X_Test_List[[i]]
  Y_Test <- YTS_pca_list[[i]]
  models <- rf_models_list[[i]]
  pca_obj <- prcomp(Y_Train_List[[i]], center = TRUE, scale. = TRUE)

  results <- compute_prediction_intervals(X_Test, Y_Test, models, pca_obj)

  mean_width_rf_list[[i]] <- results$mean_width_rf_percentage
  coverage_percentage_list[[i]] <- results$coverage_percentage

  print(paste("Mean Width of Prediction Intervals for Dataset", i, ":", results$mean_width_rf_percentage))
  print(paste("Coverage Percentage for Dataset", i, ":", results$coverage_percentage))
}

#function to produce observed vs predicted plots
plot_predictions_and_errors <- function(X_Test, Y_Test, models, pca_obj, dataset_name) {

  predictions <- matrix(NA, nrow = nrow(X_Test), ncol = length(models))
  for (j in 1:length(models)) {
    model <- models[[j]]
    predictions[, j] <- predict(model, newdata = X_Test)
  }


  predicted_Y_Test <- predictions %*% t(pca_obj$rotation[, 1:ncol(predictions)])
  predicted_Y_Test <- scale(predicted_Y_Test, center = FALSE, scale = 1 / pca_obj$scale)
  predicted_Y_Test <- scale(predicted_Y_Test, center = -pca_obj$center, scale = FALSE)


  mean_Y_Test <- colMeans(Y_Test)
  mean_predicted_Y_Test <- colMeans(predicted_Y_Test)
  mean_errors <- mean_predicted_Y_Test - mean_Y_Test


  percentage_errors <- (mean_errors / mean_Y_Test) * 100


  df_observed_predicted <- data.frame(
    Years = years,
    Observed = mean_Y_Test,
    Predicted = mean_predicted_Y_Test
  )

  df_errors <- data.frame(
    Years = years,
    Errors = percentage_errors
  )


  observed_vs_predicted_plot <- ggplot(df_observed_predicted, aes(x = Years)) +
    geom_line(aes(y = Observed, color = "Observed")) +
    geom_line(aes(y = Predicted, color = "Predicted")) +
    labs(x = "Years", y = paste(dataset_name, "(Test Data)"),
         title = paste(dataset_name, "RF Ensemble Mean")) +
    scale_color_manual(values = c("Observed" = "black", "Predicted" = "red")) +
    scale_y_continuous(limits = range(c(mean_Y_Test, mean_predicted_Y_Test))) +
    theme_minimal()


  errors_plot <- ggplot(df_errors, aes(x = Years, y = Errors)) +
    geom_line(color = "black") +
    labs(x = "Years", y = "Percentage Error",
         title = paste(dataset_name, "RF Error Ensemble Mean (Percentage)")) +
    scale_y_continuous(limits = c(-50, 50)) +
    theme_minimal()

  return(list(observed_vs_predicted = observed_vs_predicted_plot, errors = errors_plot))
}

#plot the data
plot_list <- list()
for (i in seq_along(rf_models_list)) {
  X_Test <- X_Test_List[[i]]
  Y_Test <- Y_Test_List[[i]]
  models <- rf_models_list[[i]]
  Data_Name <- Data_Names_list[[i]]
  pca_obj <- prcomp(Y_Train_List[[i]], center = TRUE, scale. = TRUE)

  plots <- plot_predictions_and_errors(X_Test, Y_Test, models, pca_obj, paste(Data_Name))
  plot_list[[i]] <- grid.arrange(
    plots$observed_vs_predicted,
    plots$errors,
    nrow = 2,
    top = paste(Data_Name, "Plots")
  )
}
```

GP

```
library(tidyverse)
library(ggplot2)
library(reshape2)
library(ggpubr)
library(caret)
library(gridExtra)
library(beepr)
library(Metrics)
library(DiceKriging)
library(rBayesianOptimization)

load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/Loop_data.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/All_gp_lists.RData")

optim_function <- list()
result <- list()
tuned_gp <- list()
time_taken_gp_list<- list()

#Bayesian optimization to tune GP models for all datasets
set.seed(123)
for (j in seq_along(YTR_pca_list)) {
  optim_function[[j]] <- list()
  result[[j]] <- list()
  tuned_gp[[j]] <- list()

  time_taken_gp_list[[j]] <- system.time({
    set.seed(123)
    for (i in 1:npcs_list[[j]]) {
      optim_function[[j]][[i]] <- function(nugget, range) {
        model <- km(formula = ~., design = X_Train_List[[j]], response = YTR_pca_list[[j]][, i],
                    covtype = "matern5_2", nugget = nugget,
                    parinit = list(range = range))
        loo_pred <- leaveOneOut.km(model, type = "UK")
        mse <- mean((loo_pred$mean - YTR_pca_list[[j]][, i])^2)
        rmse <- sqrt(mse)
        list(Score = -rmse, Pred = 0)
      }

      bounds <- list(nugget = c(1e-10, 1), range = c(1e-6, 1))
      set.seed(123)
      result[[j]][[i]] <- BayesianOptimization(
        FUN = optim_function[[j]][[i]], bounds = bounds,
        init_points = 10, n_iter = 3,
        acq = "ucb", kappa = 2.576, eps = 0.0,
        verbose = FALSE
      )

      tuned_gp[[j]][[i]] <- km(
        formula = ~., design = X_Train_List[[j]], response = YTR_pca_list[[j]][, i],
        covtype = "matern5_2", nugget = result[[j]][[i]]$Best_Par[["nugget"]],
        parinit = list(range = result[[j]][[i]]$Best_Par[["range"]]),
        control = list(trace = FALSE)
      )
    }
  })
```

```r
}
beep(3)

all_gp_predictions_list <- list()
all_gp_pred_values_list <- list()
all_gp_pred_std_error_list <- list()

#acquire GP predictions and standard error
for (j in seq_along(tuned_gp)) {
  all_gp_predictions_list[[j]] <- list()
  all_gp_pred_values_list[[j]] <- matrix(NA, nrow = nrow(X_Test_List[[j]]), ncol = length(tuned_gp[[j]]))
  all_gp_pred_std_error_list[[j]] <- matrix(NA, nrow = nrow(X_Test_List[[j]]), ncol = length(tuned_gp[[j]]))

  for (i in seq_along(tuned_gp[[j]])) {
    gp_model <- tuned_gp[[j]][[i]]
    all_gp_predictions_list[[j]][[i]] <- predict.km(gp_model, newdata = X_Test_List[[j]], type = "UK")
    all_gp_pred_values_list[[j]][, i] <- all_gp_predictions_list[[j]][[i]]$mean
    all_gp_pred_std_error_list[[j]][, i] <- all_gp_predictions_list[[j]][[i]]$sd
  }

}

gp_plot_data_list <- list()

#setup GP results for plotting pca space
for (j in seq_along(all_gp_pred_values_list)) {
  plot_data_list_j <- list()

  for (i in 1) {
    gp_predicted <- all_gp_pred_values_list[[j]][, i]
    gp_actual <- YTS_pca_list[[j]][, i]
    std_error <- all_gp_pred_std_error_list[[j]][, i]
    plot_data <- data.frame(
      Observed = as.numeric(gp_actual),
      Predicted = as.numeric(gp_predicted),
      Std_Error = std_error,
      Dataset = paste("Dataset", j),
      Component = paste("Component", i),
      Method = c("GaussianProcess")
    )
    gp_plot_data_list[[length(gp_plot_data_list) + 1]] <- plot_data
  }
}

gp_rmse_list<- list()
#get RMSEs for all GP models pca space
for (j in seq_along(all_gp_pred_values_list)) {
  actual_values <- YTS_pca_list[[j]][, 1]
  predicted_values <- all_gp_pred_values_list[[j]][, 1]
  rmse <- sqrt(mean((actual_values - predicted_values)^2, na.rm = TRUE))
  gp_rmse_list[[j]] <- rmse
  print(paste("RMSE for Dataset", j, "Component 1:", rmse))
}
```

```r
gp_test_list <- list()
pred_intervals_matrix_gp_list <- list()
mean_width_gp_list <- list()
coverage_percentage_gp_list<- list()

#acquire mean widths and coverage percentage
for (j in seq_along(all_gp_pred_values_list)) {
  pca_rotation <- pca_list[[j]]$rotation[, 1:npcs_list[[j]]]

  gp_Test <- all_gp_pred_values_list[[j]] %*% t(pca_rotation)

  gp_Test <- scale(gp_Test, center = FALSE, scale = 1 / pca_list[[j]]$scale)
  gp_Test <- scale(gp_Test, center = -pca_list[[j]]$center, scale = FALSE)
  #mean widths
  lower_quantile_gp <- apply(gp_Test, 1, quantile, probs = 0.05)
  upper_quantile_gp <- apply(gp_Test, 1, quantile, probs = 0.95)

  pred_intervals_matrix_gp <- data.frame(Lower_Quantile = lower_quantile_gp,
                                         Upper_Quantile = upper_quantile_gp)
  pred_intervals_matrix_gp_list[[j]] <- pred_intervals_matrix_gp

  interval_widths_gp <- pred_intervals_matrix_gp$Upper_Quantile
  - pred_intervals_matrix_gp$Lower_Quantile

  y_range <- max(YTS_pca_list[[j]]) - min(YTS_pca_list[[j]])
  normalized_interval_widths_gp <- interval_widths_gp / y_range

  mean_width_gp <- mean(normalized_interval_widths_gp, na.rm = TRUE)
  mean_width_gp_percentage <- mean_width_gp * 100
  mean_width_gp_list[[j]] <- mean_width_gp_percentage
  #coverage
  within_interval <- gp_Test[[j]] >= pred_intervals_matrix_gp$Lower_Quantile
  & gp_Test[[j]] <= pred_intervals_matrix_gp$Upper_Quantile
  coverage_percentage <- mean(within_interval) * 100
  coverage_percentage_gp_list[[j]]<- coverage_percentage

  print(paste("Mean Width of Prediction Intervals (GP) for Dataset", j, ":", mean_width_gp_percentage))
  print(paste("Coverage Percentage for Dataset", j, ":", coverage_percentage))
}

Data_Names_list<- c("BareSoilFrac","C3Plants","C4Plants","CSoil","CVeg","fHarvest",
                    "CumFLuc","Lai","CumNBP","NPP","Rh_land_sum","ShrubFrac","TreeFrac")

#function for getting plot data adapted from  RF for GP
plot_predictions_and_errors_gp <- function(X_Test, Y_Test, gp_pred_values, pca_obj, dataset_name) {

  predicted_Y_Test <- gp_pred_values %*% t(pca_obj$rotation[, 1:ncol(gp_pred_values)])
  predicted_Y_Test <- scale(predicted_Y_Test, center = FALSE, scale = 1 / pca_obj$scale)
  predicted_Y_Test <- scale(predicted_Y_Test, center = -pca_obj$center, scale = FALSE)

  mean_Y_Test <- colMeans(Y_Test)
  mean_predicted_Y_Test <- colMeans(predicted_Y_Test)
  mean_errors <- mean_predicted_Y_Test - mean_Y_Test
  percentage_errors <- (mean_errors / mean_Y_Test) * 100

  df_observed_predicted <- data.frame(
    Years = years,
    Observed = mean_Y_Test,
    Predicted = mean_predicted_Y_Test
  )

  df_errors <- data.frame(
    Years = years,
    Errors = percentage_errors
  )

  observed_vs_predicted_plot <- ggplot(df_observed_predicted, aes(x = Years)) +
    geom_line(aes(y = Observed, color = "Observed")) +
    geom_line(aes(y = Predicted, color = "Predicted")) +
    labs(x = "Years", y = "Test Data",
         title = paste(dataset_name, "GP Ensemble Mean")) +
    scale_color_manual(values = c("Observed" = "black", "Predicted" = "red")) +
    scale_y_continuous(limits = range(c(mean_Y_Test, mean_predicted_Y_Test))) +
    theme_minimal()+
    theme(axis.title = element_text(size = 8),
          plot.title = element_text(size = 12))

  errors_plot <- ggplot(df_errors, aes(x = Years, y = Errors)) +
    geom_line(color = "black") +
    labs(x = "Years", y = "Percentage Error",
         title = paste(dataset_name, "GP Error Mean")) +
    theme_minimal()+
    theme(axis.title = element_text(size = 8),
          plot.title = element_text(size = 12))

  return(list(observed_vs_predicted = observed_vs_predicted_plot, errors = errors_plot))
}

observed_vs_predicted_gp_plots <- list()
errors_gp_plots <- list()

# Create the plots
for (i in seq_along(all_gp_pred_values_list)) {
  X_Test <- X_Test_List[[i]]
  Y_Test <- Y_Test_List[[i]]
  gp_pred_values <- all_gp_pred_values_list[[i]]
  Data_Name <- Data_Names_list[[i]]
  pca_obj <- pca_list[[i]]

  plots <- plot_predictions_and_errors_gp(X_Test, Y_Test, gp_pred_values, pca_obj, paste(Data_Name))

  observed_vs_predicted_gp_plots[[i]] <- plots$observed_vs_predicted
  errors_gp_plots[[i]] <- plots$errors
}
```

```
# Arrange plots into grid
observed_vs_predicted_gp_grid <- ggarrange(
  plotlist = observed_vs_predicted_gp_plots,
  ncol = 5,
  nrow = 3,
  common.legend = TRUE
)
print(observed_vs_predicted_gp_grid)

#plot grid for errors
errors_gp_grid <- ggarrange(
  plotlist = errors_gp_plots,
  ncol = 5,
  nrow = 3,
  common.legend = TRUE
)
print(errors_gp_grid)
```

XGBoost

```
library(tidyverse)
library(ggplot2)
library(reshape2)
library(ggpubr)
library(caret)
library(gridExtra)
library(xgboost)
library(Metrics)

load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/Loop_data.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/functions.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/All_xgb_lists.RData")

time_taken_xgb_list<- list()
xgb_models_list<- list()

#train and tune XGBoost models with caret grid search
for (i in seq_along(YTR_pca_list)) {
  YTR_pca <- YTR_pca_list[[i]]
  X_Train <- X_Train_List[[i]]

  models <- list()
  time_taken_xgb <- system.time({
    set.seed(123)
    for (j in 1:ncol(YTR_pca)) {
      models[[j]] <- train(x = X_Train, y = YTR_pca[, j],
                          method = "xgbTree",
                          objective = "reg:squarederror",
                          trControl = trainControl(method = "cv", number = 5,
                                                   verboseIter = F),
                          tuneGrid = expand.grid(nrounds = c(50,100,150),
                                                  eta = c(0.1,0.2,0.3),
                                                  max_depth = c(2,4,6),
                                                  gamma = c(0,0.1,1),
                                                  colsample_bytree = c(0.3,0.6,1),
                                                  subsample = c(0.3,0.6,1),
                                                  min_child_weight = c(0,1,3)))

    }
  })
  xgb_models_list[[i]]<- models
  time_taken_xgb_list[[i]]<- time_taken_xgb

  cat("Finished processing YTR_pca_list element", i, "\n")
}

all_predictions_xgb <- list()
all_actual_xgb <- list()

#get XGB predictions
for (i in seq_along(xgb_models_list)) {
  X_Test <- X_Test_List[[i]]
  Y_Test <- YTS_pca_list[[i]]
  xgb_models <- xgb_models_list[[i]]

  predictions_xgb <- matrix(NA, nrow = nrow(X_Test), ncol = length(xgb_models))
  for (j in 1:length(xgb_models)) {
    xgb_model <- xgb_models[[j]]
    predictions_xgb[, j] <- predict(xgb_model, newdata = X_Test)
  }

  all_predictions_xgb[[i]] <- predictions_xgb
  all_actual_xgb[[i]] <- Y_Test
}

xgb_plot_data_list <- list()

#XGB pca prediction vs observed plots
for (i in seq_along(all_predictions_xgb)) {
  predictions <- all_predictions_xgb[[i]]
  actual <- all_actual_xgb[[i]]

  j <- 1
  if (ncol(predictions) >= j) {
    xgb_predicted <- predictions[, j]
    xgb_actual <- actual[, j]

    residuals <- xgb_actual - xgb_predicted
    std_error <- rep(sd(residuals), length(xgb_predicted))

    xgb_plot_data <- data.frame(
      Observed = as.numeric(xgb_actual),
      Predicted = as.numeric(xgb_predicted),
      Std_Error = std_error,
      Dataset = paste("Dataset", i),
      Component = paste("Component", j),
      Method = c("XGBoost")
    )

    xgb_plot_data_list[[length(xgb_plot_data_list) + 1]] <- xgb_plot_data
  }
}

xgb_rmse_list<- list()

#calculate XGB RMSEs pca space
for (i in seq_along(xgb_models_list)) {
  X_Test <- X_Test_List[[i]]
  Y_Test <- YTS_pca_list[[i]]
  xgb_models <- xgb_models_list[[i]]

  predictions_xgb <- predict(xgb_models[[1]], newdata = X_Test)

  actual_xgb <- Y_Test[, 1]
  rmse_value_xgb <- rmse(actual_xgb, predictions_xgb)

  xgb_rmse_list[[i]] <- rmse_value_xgb

  print(paste("RMSE for Dataset", i, "Component 1:", rmse_value_xgb))
}

mean_width_xgb_list <- list()
coverage_percentage_xgb_list <- list()

#get mean width and coverage for XGB
for (i in seq_along(xgb_models_list)) {
  X_Test <- X_Test_List[[i]]
  Y_Test <- YTS_pca_list[[i]]
  models <- xgb_models_list[[i]]
  pca_obj <- prcomp(Y_Train_List[[i]], center = TRUE, scale. = TRUE)

  results <- compute_prediction_intervals(X_Test, Y_Test, models, pca_obj)
  #note states "rf" this is due to using the same function as used for random
  #forest as found to work for both rf and XGB methods
  mean_width_xgb_list[[i]] <- results$mean_width_rf_percentage
  coverage_percentage_xgb_list[[i]] <- results$coverage_percentage

  print(paste("Mean Width of Prediction Intervals for Dataset", i, ":", results$mean_width_rf_percentage))
  print(paste("Coverage Percentage for Dataset", i, ":", results$coverage_percentage))
}

#function slightly altered from rf version to use XGB results
plot_predictions_and_errors_xgb <- function(X_Test, Y_Test, models, pca_obj, dataset_name) {
  predictions <- matrix(NA, nrow = nrow(X_Test), ncol = length(models))
  for (j in 1:length(models)) {
    model <- models[[j]]
    predictions[, j] <- predict(model, newdata = X_Test)
  }

  predicted_Y_Test <- predictions %*% t(pca_obj$rotation[, 1:ncol(predictions)])
  predicted_Y_Test <- scale(predicted_Y_Test, center = FALSE, scale = 1 / pca_obj$scale)
  predicted_Y_Test <- scale(predicted_Y_Test, center = -pca_obj$center, scale = FALSE)

  mean_Y_Test <- colMeans(Y_Test)
  mean_predicted_Y_Test <- colMeans(predicted_Y_Test)
  mean_errors <- mean_predicted_Y_Test - mean_Y_Test

  percentage_errors <- (mean_errors / mean_Y_Test) * 100

  df_observed_predicted <- data.frame(
    Years = years,
    Observed = mean_Y_Test,
    Predicted = mean_predicted_Y_Test
  )

  df_errors <- data.frame(
    Years = years,
    Errors = percentage_errors
  )

  observed_vs_predicted_plot <- ggplot(df_observed_predicted, aes(x = Years)) +
    geom_line(aes(y = Observed, color = "Observed")) +
    geom_line(aes(y = Predicted, color = "Predicted")) +
    labs(x = "Years", y = "Test Data",
         title = paste(dataset_name, "XGB Ensemble Mean")) +
    scale_color_manual(values = c("Observed" = "black", "Predicted" = "red")) +
    scale_y_continuous(limits = range(c(mean_Y_Test, mean_predicted_Y_Test))) +
    theme_minimal()+
    theme(axis.title = element_text(size = 8),
          plot.title = element_text(size = 12))

  errors_plot <- ggplot(df_errors, aes(x = Years, y = Errors)) +
    geom_line(color = "black") +
    labs(x = "Years", y = "Percentage Error",
         title = paste(dataset_name, "XGB Error Mean")) +
    theme_minimal()+
    theme(axis.title = element_text(size = 8),
          plot.title = element_text(size = 12))

  return(list(observed_vs_predicted = observed_vs_predicted_plot, errors = errors_plot))
}

observed_vs_predicted_plots_xgb <- list()
errors_plots_xgb <- list()

#acquire the plots
for (i in seq_along(xgb_models_list)) {
  Data_Name <- Data_Names_list[[i]]
  X_Test <- X_Test_List[[i]]
  Y_Test <- Y_Test_List[[i]]
  models <- xgb_models_list[[i]]
  pca_obj <- prcomp(Y_Train_List[[i]], center = TRUE, scale. = TRUE)

  plots <- plot_predictions_and_errors_xgb(X_Test, Y_Test, models, pca_obj, paste(Data_Name))

  observed_vs_predicted_plots_xgb[[i]] <- plots$observed_vs_predicted
  errors_plots_xgb[[i]] <- plots$errors
}

#put plots in grid
observed_vs_predicted_grid_xgb <- ggarrange(
  plotlist = observed_vs_predicted_plots_xgb,
  ncol = 5,
  nrow = 3,
  common.legend = TRUE
)
print(observed_vs_predicted_grid_xgb)

#error plots grid
errors_grid_xgb <- ggarrange(
  plotlist = errors_plots_xgb,
  ncol = 5,
  nrow = 3,
  common.legend = TRUE
)
print(errors_grid_xgb)
```

Visualisations

```
library(tidyverse)
library(ggplot2)
library(reshape2)
library(randomForest)
library(ggpubr)
library(caret)
library(gridExtra)
library(xgboost)
library(DiceKriging)
library(rBayesianOptimization)
library(beepr)
library(Metrics)
library(gt)

load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/Loop_data.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/functions.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/All_xgb_lists.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/All_rf_lists.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/All_gp_lists.RData")
load("C:/Users/tjsla/OneDrive/Desktop/Masters/Dissertation/gp_coverage.RData")

#convert proc time object to dataframe
```

```r
proc_time_to_elapsed_df <- function(proc_time_list) {
  elapsed_times <- sapply(proc_time_list, function(pt) {
    pt['elapsed']
  })

  data.frame(elapsed_time = elapsed_times)
}

time_taken_gp_df <- proc_time_to_elapsed_df(time_taken_gp_list)
time_taken_rf_df <- proc_time_to_elapsed_df(time_taken_rf_list)
time_taken_xgb_df <- proc_time_to_elapsed_df(time_taken_xgb_list)

#divide by 60 to get minutes as proc time is recorded in seconds
time_taken_gp_df<- data.frame(ComputationalTime = time_taken_gp_df$elapsed_time/60,
                              Dataset = Data_Names_list, Method = c("GaussianProcess"))
time_taken_rf_df<- data.frame(ComputationalTime = time_taken_rf_df$elapsed_time/60,
                              Dataset = Data_Names_list, Method = c("RandomForest"))
time_taken_xgb_df<- data.frame(ComputationalTime = time_taken_xgb_df$elapsed_time/60,
                               Dataset = Data_Names_list, Method = c("XGBoost"))

#combine comp time datasets for plotting
comp_times<- data.frame(rbind(time_taken_gp_df,time_taken_rf_df,time_taken_xgb_df))

#produce comp time plot
colours <- c("GaussianProcess" = "#FFC107", "RandomForest" = "#50C878", "XGBoost" = "#0F52BA")
ggplot(comp_times, aes(x = Dataset, y = ComputationalTime, fill = Method)) +
  geom_bar(stat = "identity", position = "dodge") +
  geom_text(aes(label = round(ComputationalTime, 1)),
            position = position_dodge(width = 0.9),
            vjust = -0.5,
            color = "black",
            size = 3.5) +
  scale_fill_manual(values = colours) +
  labs(title = "Computational Time by Method and Dataset",
       x = "Dataset",
       y = "Computational Time (Minutes)",
       fill = "Method") +
  theme_minimal()+
  annotate("text", x = 13,y = 150, label = c("RF Cum Run Time: 140.5"), hjust = 1, vjust = 0,
           color = "#50C878", size = 4)+
  annotate("text", x = 13,y = 142, label = c("GP Cum Run Time: 425.1"), hjust = 1, vjust = 0,
           color = "#FFC107", size = 4)+
  annotate("text", x = 13,y = 134, label = c("XGB Cum Run Time: 835.8"), hjust = 1, vjust = 0,
           color = "#0F52BA", size = 4)

#combine the plot data previously produced
combined_data_list<- list()
for (i in seq_along(gp_plot_data_list)){
  gp_plot_data<- gp_plot_data_list[[i]]
  rf_plot_data<- rf_plot_data_list[[i]]
  xgb_plot_data<- xgb_plot_data_list[[i]]

  combined_data <- rbind(rf_plot_data, xgb_plot_data, gp_plot_data)
  combined_data_list[[i]]<- combined_data
}

#combine the RMSEs previously calculated
rmses_list<- list()
for (i in seq_along(gp_rmse_list)){
  gp_rmse<- gp_rmse_list[[i]]
  rf_rmse<- rf_rmse_list[[i]]
  xgb_rmse<- xgb_rmse_list[[i]]

  gp_rmse<- as.character(round(gp_rmse,2))
  rf_rmse<- as.character(round(rf_rmse,2))
  xgb_rmse<- as.character(round(xgb_rmse,2))

  rmses_list[[i]]<- c(gp_rmse,rf_rmse,xgb_rmse)
}

plots <- vector("list", length(combined_data_list))

#store the pca space plots
for (i in seq_along(combined_data_list)) {
  combined_data <- combined_data_list[[i]]

  rmse_values <- rmses_list[[i]]
  gp_rmse <- rmse_values[1]
  rf_rmse <- rmse_values[2]
  xgb_rmse <- rmse_values[3]
  dataset<- Data_Names_list[i]

  plots[[i]] <- ggplot(combined_data, aes(x = Observed, y = Predicted, color = Method, shape = Method)) +
    geom_point(alpha = 0.75) +
    geom_errorbar(aes(ymin = Predicted - Std_Error, ymax = Predicted + Std_Error),
                  width = 0.2, alpha = 0.5) +
    geom_abline(intercept = 0, slope = 1, color = "black") +
    labs(x = "Observed", y = "Predicted", title = paste("Observed vs Predicted PC1", dataset)) +
    theme_minimal() +
    theme(legend.position = "right") +
    scale_color_manual(values = colours) +
    annotate("text", x = 35, y = -16, label = paste("GP RMSE =", gp_rmse), hjust = 1, vjust = 0,
             color = "#FFC107", size = 4) +
    annotate("text", x = 35, y = -21, label = paste("RF RMSE =", rf_rmse), hjust = 1, vjust = 0,
             color = "#50C878", size = 4) +
    annotate("text", x = 35, y = -26, label = paste("XGBoost RMSE =", xgb_rmse), hjust = 1, vjust = 0,
             color = "#0F52BA", size = 4)
}


ncol <- 2
nrow <- 2

#put plots in a grid
create_grid <- function(plots) {
  grid.arrange(grobs = plots, ncol = ncol)
```

```
}

grid1 <- create_grid(plots[1:4])
grid2 <- create_grid(plots[5:8])
grid3 <- create_grid(plots[9:12])
grid4 <- grid.arrange(grobs = plots[13], ncol = 2,nrow = 2)

all_predictions_rf_orig <- list()
all_actual_rf_orig <- list()

#reobtain predictions for RF
for (i in seq_along(all_predictions_rf)) {
  pca_obj <- prcomp(Y_Train_List[[i]], center = TRUE, scale. = TRUE)
  predictions_pca <- all_predictions_rf[[i]]
  actual_y_test <- Y_Test_List[[i]]

  predicted_Y_Test <- predictions_pca %*% t(pca_obj$rotation[, 1:npcs_list[i]])
  predicted_Y_Test <- scale(predicted_Y_Test, center = FALSE, scale = 1/pca_obj$scale)
  predicted_Y_Test <- scale(predicted_Y_Test, center = -pca_obj$center, scale = FALSE)

  all_predictions_rf_orig[[i]] <- as.data.frame(predicted_Y_Test)
  all_actual_rf_orig[[i]] <- as.data.frame(actual_y_test)
}

#calculate ranks for rank histogram function
calculate_ranks <- function(observation, predictions) {

  combined <- c(predictions, observation)
  ranks <- rank(combined)
  return(ranks[1:length(predictions)])
}

all_ranks_rf <- list()
#calculate the ranks
for (i in seq_along(all_predictions_rf_orig)) {
  predictions <- all_predictions_rf_orig[[i]]
  actual <- all_actual_rf_orig[[i]]


  ranks_matrix <- t(apply(predictions, 1, function(pred) {
    sapply(1:ncol(predictions), function(j) calculate_ranks(actual[, j], pred[j]))
  }))

  all_ranks_rf[[i]] <- ranks_matrix
}
#function to normalize ranks
normalize_ranks <- function(ranks) {
  min_rank <- min(ranks, na.rm = TRUE)
  max_rank <- max(ranks, na.rm = TRUE)

  if (min_rank == max_rank) {
    return(rep(0.5, length(ranks)))
  }
  scaled_ranks <- (ranks - min_rank) / (max_rank - min_rank)
  return(scaled_ranks)
}

rank_plots_list_rf <- list()
#setup ranks for plotting
for (i in seq_along(all_ranks_rf)) {
  ranks_matrix <- all_ranks_rf[[i]]

  ranks_vector <- as.vector(ranks_matrix)

  normalized_ranks <- normalize_ranks(ranks_vector)


  rank_data <- data.frame(ranks = normalized_ranks)
  dataset <- Data_Names_list[i]

  p <- ggplot(rank_data, aes(x = ranks)) +
    geom_histogram(bins = 10, color = "black", fill = "blue") +
    labs(title = paste("RF Rank Histogram", dataset),
         x = "Normalized Rank",
         y = "Frequency") +
    theme_minimal()

  rank_plots_list_rf[[i]] <- p
}
#put in a grid
grid.arrange(grobs = rank_plots_list_rf, ncol = 4)

all_predictions_xgb_orig <- list()
all_actual_xgb_orig <- list()

for (i in seq_along(all_predictions_xgb)) {
  pca_obj <- prcomp(Y_Train_List[[i]], center = TRUE, scale. = TRUE)
  predictions_pca <- all_predictions_xgb[[i]]
  actual_y_test <- Y_Test_List[[i]]

  predicted_Y_Test <- predictions_pca %*% t(pca_obj$rotation[, 1:npcs_list[i]])
  predicted_Y_Test <- scale(predicted_Y_Test, center = FALSE, scale = 1/pca_obj$scale)
  predicted_Y_Test <- scale(predicted_Y_Test, center = -pca_obj$center, scale = FALSE)

  all_predictions_xgb_orig[[i]] <- as.data.frame(predicted_Y_Test)
  all_actual_xgb_orig[[i]] <- as.data.frame(actual_y_test)
}

all_ranks_xgb<- list()
#calculate XGBoost ranks
for (i in seq_along(all_predictions_xgb_orig)) {
  predictions <- all_predictions_xgb_orig[[i]]
  actual <- all_actual_xgb_orig[[i]]

  ranks_matrix <- t(apply(predictions, 1, function(pred) {
    sapply(1:ncol(predictions), function(j) calculate_ranks(actual[, j], pred[j]))
  }))

  all_ranks_xgb[[i]] <- ranks_matrix
}

rank_plots_list_xgb <- list()
#plot XGB ranls
for (i in seq_along(all_ranks_xgb)) {
  ranks_matrix <- all_ranks_xgb[[i]]

  ranks_vector <- as.vector(ranks_matrix)

  normalized_ranks <- normalize_ranks(ranks_vector)


  rank_data <- data.frame(ranks = normalized_ranks)
  dataset <- Data_Names_list[i]

  p <- ggplot(rank_data, aes(x = ranks)) +
    geom_histogram(bins = 10, color = "black", fill = "blue") +
    labs(title = paste("XGB Rank Histogram", dataset),
         x = "Normalized Rank",
         y = "Frequency") +
    theme_minimal()


  rank_plots_list_xgb[[i]] <- p
}
grid.arrange(grobs = rank_plots_list_xgb, ncol = 4)

all_predictions_gp_orig <- list()
all_actual_gp_orig <- list()

for (i in seq_along(all_gp_pred_values_list)) {
  pca_obj <- prcomp(Y_Train_List[[i]], center = TRUE, scale. = TRUE)
  predictions_pca <- all_gp_pred_values_list[[i]]
  actual_y_test <- Y_Test_List[[i]]

  predicted_Y_Test <- predictions_pca %*% t(pca_obj$rotation[, 1:npcs_list[i]])
  predicted_Y_Test <- scale(predicted_Y_Test, center = FALSE, scale = 1/pca_obj$scale)
  predicted_Y_Test <- scale(predicted_Y_Test, center = -pca_obj$center, scale = FALSE)

  all_predictions_gp_orig[[i]] <- as.data.frame(predicted_Y_Test)
  all_actual_gp_orig[[i]] <- as.data.frame(actual_y_test)
}

all_ranks_gp<- list()
#calculate GP ranks
for (i in seq_along(all_predictions_gp_orig)) {
  predictions <- all_predictions_gp_orig[[i]]
  actual <- all_actual_gp_orig[[i]]

  ranks_matrix <- t(apply(predictions, 1, function(pred) {
    sapply(1:ncol(predictions), function(j) calculate_ranks(actual[, j], pred[j]))
  }))

  all_ranks_gp[[i]] <- ranks_matrix
}

rank_plots_list_gp <- list()
#plot the ranks
for (i in seq_along(all_ranks_gp)) {
  ranks_matrix <- all_ranks_gp[[i]]

  ranks_vector <- as.vector(ranks_matrix)

  normalized_ranks <- normalize_ranks(ranks_vector)

  rank_data <- data.frame(ranks = normalized_ranks)
  dataset <- Data_Names_list[i]

  p <- ggplot(rank_data, aes(x = ranks)) +
    geom_histogram(bins = 10, color = "black", fill = "blue") +
    labs(title = paste("GP Rank Histogram", dataset),
         x = "Normalized Rank",
         y = "Frequency") +
    theme_minimal()


  rank_plots_list_gp[[i]] <- p
}
grid.arrange(grobs = rank_plots_list_gp, ncol = 4)

#setup mean width lists for making tables
mw_gp<- unlist(mean_width_gp_list)
mw_gp<- data.frame(Mean_Width_GP = mw_gp)

mw_rf<- unlist(mean_width_rf_list)
mw_rf<- data.frame(Mean_Width_RF = mw_rf)

mw_xgb<- unlist(mean_width_xgb_list)
mw_xgb<- data.frame(Mean_Width_XGB = mw_xgb)
#dataframe for putting in table
mw_df<- data.frame(cbind(mw_gp,mw_rf,mw_xgb))
mw_df<- data.frame(sapply(mw_df, function(x) round(x, 2)))
mw_df<- data.frame(mw_df, Dataset=Data_Names_list)

gt_width <- gt(mw_df)
#function to highlight minimum value/s for each row
highlight_min_in_row <- function(data, gt_table) {

  for (i in 1:nrow(data)) {

    min_value <- min(data[i, 1:3])

    min_cols <- which(data[i, 1:3] == min_value)
    gt_table <- gt_table %>%
      tab_style(
        style = cell_fill(color = "lightgreen"),
        locations = cells_body(
          columns = all_of(names(data)[min_cols]),
          rows = i
        )
      )
  }
  return(gt_table)
}

#finalise the table
gt_width <- highlight_min_in_row(mw_df, gt_width) %>%
  tab_header(title = "Percentage uncertainty (CI Widths)")

gt_width

input_desc<- data.frame(Input_Parameter = c("alpha", "a_wl", "bio_hum_cn", "b_wl", "dcatch_dlai",
                                            "dqcrit_io", "dz0v_dh", "f0", "fd", "g_area", "g_root",
                                            "g_wood", "gs_nvg", "hw_sw", "kaps_roth", "knl", "lai_max",
                                            "lai_min", "lma", "n_inorg_turnover", "nmass", "nr", "retran_l",
                                            "retran_r", "r_grow", "rootd_ft", "sigl", "sorp", "tleaf_of",
                                            "tlow", "tupp", "l_vg_soil"),
                        Description = c("Quantum efficiency", "Allometric coefficient - relates woody biomass to lai",
                                        "Carbon to Nitrogen ratio microbial biomass and long-lived humidified pools",

                                        "Allometric exponent - relates woody biomass to lai", "Rate of change - canopy capacit

                                        "Critical humidity deficit", "Rate of Change - vegetation roughness", "Maximum ratio -

                                        "Scale factor - Dark respiration", "Disturbance rate", "Turnover rate - root biomass",

                                        "Turnover rate - woody biomass", "Surface conductance", "Ratio - nitrogen stem to nit

                                        "Soil respiration rate - RothC submodel", "Decay of nitrogen through canopy - function

                                        "Maximum lai", "Minimum lai", "Leaf mass per unit area", "Lifetime of inorganic nitro

                                        "Top leaf nitrogen content per unit mass", "Root nitrogen concentration", "Fraction of

                                        "Fraction of retranslocated root nitrogen", "Growth respiration fraction", "Root depth"

                                        "Specific density of leaf carbon", "Leaching of inorganic nitrogen through soil profile",

                                        "Temperature bellow which leaves are dropped", "Lower temperature for photosynthesi

                                        "Upper temperature for photosynthesis", "Switch for van Genuchten soil hydraulic mod

                        Reference = c(rep("McNeall et al, 2024", 32)))

#table for input descriptions
gt_input<- gt(input_desc)
```

```
gt_input<- gt_input %>%
  tab_header(title = "Table 1 Input Parameter Descriptions") %>%
  tab_style(
    style = cell_borders(sides = "left", color = "black", weight = px(1)),
    locations = list(
      cells_body(columns = everything()),
      cells_column_labels(columns = everything())
    )
  )

gt_input

output_desc<- data.frame(Output_Variable = c("BareSoilFrac","C3Plants", "C4Plants","CSoil","CVeg","FHarvest"
                                             "NBP","NPP","RhLndSum","ShrubFrac","TreeFrac"),
                         Description = c("Proportion of land covered by bare soil",
                                         "Proportion of land covered by C3 photosynthetic pathway plants",

                                         "Rroportion of land covered by C4 photosynthetic pathway plants", "Amount of carbo

                                         "Amount of carbon in vegetation", "Amount of carbon being passed into the atmosphe

                                         "Net carbon flux of carbon to the atmosphere resulting from land use changes e.g. defo

                                         "Leaf area index obtained by total upper leaf surface area of vegetation/horizontal sur

                                         "Net biome production", "Net primary production","Total heterotrophic respiration or

                                         "Proportion of land covered by shrubs","Proportion of land covered by trees"),

                         Reference = c(rep("Centre for Environmental Data Analysis, 2024",13)))

#table for output descriptions
gt_output<- gt(output_desc)
gt_output<- gt_output %>%
  tab_header(title = "Table 2 Output Variable Descriptions") %>%
  tab_style(
    style = cell_borders(sides = "left", color = "black", weight = px(1)),
    locations = list(
      cells_body(columns = everything()),
      cells_column_labels(columns = everything())
    )
  ) %>%
  cols_width(
    c(Output_Variable) ~ "26.5mm",
    c(Description) ~ "53mm",
    c(Reference) ~ "53mm"
  )


gt_output

#table for principal components
pc_df<- data.frame(DataSet = Data_Names_list,
                   "Number of Principal Compents"=npcs_list)
gt_pc<- gt(pc_df)
gt_pc<- gt_pc %>%
  tab_header(title = "Table 1 Number of PCs used for each dataset")

gt_pc

#acquire coverage percentages and put into a dataframe
cov_gp<- unlist(coverage_percentage_gp_list)
cov_gp<- data.frame(Coverage_Percentage_GP = cov_gp)

cov_rf<- unlist(coverage_percentage_list)
cov_rf<- data.frame(Coverage_Percentage_RF = cov_rf)

cov_xgb<- unlist(coverage_percentage_xgb_list)
cov_xgb<- data.frame(Coverage_Percentage_XGB = cov_xgb)

cov_df<- data.frame(cbind(cov_gp,cov_rf,cov_xgb))
cov_df<- data.frame(sapply(cov_df, function(x) round(x, 2)))
cov_df<- data.frame(cov_df, Dataset=Data_Names_list)

gt_cov <- gt(cov_df)

#function to highlight maximum value instead of minimum
highlight_max_in_row <- function(data, gt_table) {

  for (i in 1:nrow(data)) {
    max_value <- max(data[i, 1:3])

    max_cols <- which(data[i, 1:3] == max_value)

    gt_table <- gt_table %>%
      tab_style(
        style = cell_fill(color = "lightgreen"),
        locations = cells_body(
          columns = all_of(names(data)[max_cols]),
          rows = i
        )
      )
  }
  return(gt_table)
}

#finalise table
gt_cov <- highlight_max_in_row(cov_df, gt_cov) %>%
  tab_header(title = "Coverage Percentage by Model and Dataset")
gt_cov
```