universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

Beyond Linear Regression: Unleashing the Predictive Power of Merging Linear Models with Tree-Based Machine Learning

verfasst von / submitted by

Marie-Louise Leopold, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2023 / Vienna, 2023

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	UA 066 840
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Psychologie UG2002
Betreut von / Supervisor:	Univ.-Prof. Dr. Frank Scharnowski, MSc
Mitbetreut von / Co-Supervisor:	Dipl.-Ing. Dr.techn. David Steyrl

Abstract

This thesis explores the benefits of combining *tree-based machine learning* methods with linear models in psychological research. Commonly used models, such as *linear regression*, fail to capture the potentially complex relationships in psychological data. In contrast, *random forests*, a *supervised machine learning* technique, used on a metric target variable makes a mean prediction based on a subsample, called a *leaf node*, based on similarities in the predictor variable space. By learning patterns from the training data and consequently assigning samples to a leaf node, the random forest can model nonlinear, complex associations. Because of the mean prediction mechanism, it cannot model linear relationships or extrapolate to data outside the training range as it cannot follow linear trends beyond the simple mean predictor.

This work analysis the benefits of combining both approaches by using linear models at the leaf nodes of *decision trees* and their subsequent use in random forests.

Four estimators – decision tree, *linear tree*, random forest, and *linear forest* – are compared on a psychological data set predicting relationship satisfaction. *Coefficients of determination* obtained from *cross-validation* measure their performances. In addition to *training* and testing the models, nested cross-validation serves to tune the *hyperparameters* of each estimator within each main cross-validation loop.

Comparative analyses include *Student's T-test*, *Welch's test* with *Nadeau-Bengio correction*, and visual exploration of the hyperparameter space.

The results show that estimators enhanced with linear models significantly outperform those without ($\propto$ = .05) and exhibit superior stability across hyperparameter values and *train-test splits*. The linear forest shows the highest performance and stability. Notably, the linear forest also exhibits improved computational efficiency compared to the random forest.

This analysis highlights the potential of integrating tree-based machine learning with linear models to advance research methodologies also in psychology.

1 Introduction

1.1 Statistical Models Used in Psychological Research Today

Linear regression models have long been established as foundational tools in psychological research, providing a versatile framework to exam relationships between variables (Austin et al., 1998; Field et al., 2012; Tabachnick et al., 2019). Because of their broad applicability and ease of interpretation, they have become ubiquitous in the field, shaping our understanding of diverse psychological phenomena, including cognitive processes, personality traits, and psychopathology (Armey & Crowther, 2008; Austin et al., 1998; Jolly & Chang, 2019). This section outlines the basic aspects of linear regression models, highlighting their applications, advantages, and shortcomings.

Linear regression models are used to examine the relationship between a dependent variable and one or more independent variables (Field et al., 2012; Tabachnick et al., 2019). The main assumption underlying these models is that the relationship between the variables is linear (Field et al., 2012). The basic equation of a linear regression, where *y* represents the dependent variable, $x_1 - x_n$ the independent variables, $b_0 - b_n$ the coefficients representing the influence of each independent variable, and $\mathcal{E}_i$ = the error term of the i^{th} sample, which accounts for the unexplained variance, is as follows:

$$y = b_0 + \sum_{k=1}^{n} b_k x_k + \mathcal{E}_i$$

(Field et al., 2012)

Linear regression models offer several advantages in psychological research (Austin et al., 1998; Gravetter & Wallnau, 2017). Firstly, they provide a quantitative measure of the relationship between variables, enabling researchers to assess the strength of the association (Field et al., 2012). Secondly, linear regression models allow for the identification of significant predictors, thus contributing to the understanding and prediction of psychological phenomena (Field et al., 2012; Gravetter & Wallnau, 2017). Additionally, linear regressions facilitate hypothesis testing and model comparison, empowering researchers to assess the significance of relationships and determine the best-fitting model for their data (Field et al., 2012; Tabachnick et al., 2019).

However, while linear regression models serve as powerful tools for exploring the relationship between variables, their linear nature limits their ability to capture the complexity inherent in the relationships between most psychological variables (Abelson, 1995; Jolly & Chang, 2019). The intricate web of cognitive, emotional, and social processes often manifests as nonlinear associations that cannot be adequately captured by linear models alone (Armey & Crowther, 2008; Ehlers, 1995). Such nonlinear relationships may involve *thresholds*, *interactions*, or complex patterns that defy simple linear trends (Abelson, 1995; Jolly & Chang, 2019). While linear models have advantages, ease of interpretation and linear *extrapolation* capabilities – the ability to use the linear model to predict data outside the training data range by following linear trends -, their inability to model nonlinear relationships limits their effectiveness in many psychological applications (Judd et al., 2011). Linear models can be an oversimplification of psychological phenomena, potentially leading to incomplete, non-generalizable or irrelevant conclusions or theories (Dwyer et al., 2018; Jolly & Chang, 2019; Rocca & Yarkoni, 2021; Westreich & Greenland, 2013).

Psychological research often seeks to uncover complex relationships between variables, such as the nonlinear effects of stress on cognitive performance or the threshold effects of social support on well-being (Gravetter & Wallnau, 2017). Because linear models fail to capture such nuances by design, leading to oversimplified representations of psychological processes, more flexible modeling approaches, such as *tree-based machine learning algorithms,* hold potential due to their ability to capture nonlinear relationships (Breiman et al., 2017; Chekroud et al., 2021; Dwyer et al., 2018; Jolly & Chang, 2019; Westreich & Greenland, 2013). These algorithms, including *decision trees* (DT), *random forests* (RF), and *gradient boosting machines*, provide powerful tools for modeling complex interactions, uncovering nonlinear patterns, and handling high-dimensional data (Hastie et al., 2009).

1.2 Tree-Based Machine Learning Models in Psychological Research

The field of psychology has seen a growing interest in the application of *machine learning* (ML) algorithms to overcome the limitations of traditional linear models (Dwyer et al., 2018). ML is a branch of *artificial intelligence* that focuses on the development and implementation of

algorithms that enable computer systems to learn and improve from data without explicit programming (Harrison, 2019; Mitchell, 1997; Raschka & Mirjalili, 2019). It is a rapidly growing field with applications in various domains, including computer vision, natural language processing, and data analytics (Dwyer et al., 2018; Mitchell, 1997). This section provides a brief and simplified overview of ML.

At its core, ML is the application of statistical and computational techniques to enable computers to learn from data and make accurate predictions or decisions (Russell et al., 2010). This process typically involves input data, a learning algorithm, and an output or task (James et al., 2013; Zhou, 2021). The input data, whether structured or unstructured, serves as the basis for *training* and *testing* the ML model (James et al., 2013; Mitchell, 1997). The ML algorithm then processes this data to discover patterns, relationships, and dependencies that are used to make predictions or perform specific tasks (Hastie et al., 2009).

ML encompasses a wide variety of approaches and algorithms (Russell et al., 2010). A common class of algorithms is *supervised learning*, where the model is trained on labeled data, meaning that each input example is associated with a corresponding output or target value (Hastie et al., 2009; James et al., 2013). *Support vector machines*, *neural networks* and tree-based models such as DTs, RFs and gradient boosted trees are examples of supervised learning algorithms (Mitchell, 1997). By learning from labeled examples, these algorithms discover patterns and make predictions on unseen data (Hastie et al., 2009).

Another class of algorithms is *unsupervised learning*, which deals with unlabeled data and aims to discover hidden patterns or structures in the data (Hastie et al., 2009; Mitchell, 1997). *Clustering* and *dimensionality reduction* are popular techniques used in unsupervised learning (Hastie et al., 2009).

Reinforcement learning is another important branch of ML, which involves training an agent to interact with an environment and learn optimal actions through a reward-based system (Sutton & Barto, 2018).

ML techniques offer several advantages, for instance the ability to handle large and complex data sets, automate decision-making processes, and uncover intricate patterns that may not be apparent to the human observer (Chekroud et al., 2021; Dwyer et al., 2018; Judd et al., 2011; Rocca & Yarkoni, 2021; Westreich & Greenland, 2013). However, challenges such as

overfitting (when a model fits to closely to the data and cannot generalize onto new unseen data), data scarcity (when the data set is too small to train and test a model), and interpretability remain important considerations in ML research and practice. (Hastie et al., 2009)

A specific "branch" of supervised learning, widely used in various domains and recently gaining popularity in psychological research, are the tree-based algorithms which include DTs and RFs (Armey & Crowther, 2008; Jolly & Chang, 2019). These algorithms construct hierarchical DTs based on features and data attributes to make predictions or decisions (Breiman et al., 2017; James et al., 2013).

DTs are fundamental tree-based algorithms that recursively split the data based on feature thresholds to create a tree structure (Breiman et al., 2017; James et al., 2013). Each internal node represents a feature and a threshold, while each leaf node corresponds to a class or predicted value (Breiman et al., 2017; James et al., 2013). DTs have the versatility to handle both categorical and numerical features, and their structure allows for easy interpretation and visual representation (Breiman et al., 2017; James et al., 2013). However, DTs are susceptible to overfitting and struggle to generalize well to new, unseen data (Breiman, 2001a; Breiman et al., 2017).

To address the limitations of DTs, RFs have been introduced as an ensemble method that combines multiple DTs (Breiman, 2001a; Hastie et al., 2009). In an RF, each tree is trained on a randomly sampled subset of the training data, and predictions are made by aggregating the predictions of the individual trees (Hastie et al., 2009). RFs mitigate the overfitting problem of decision trees, improve prediction accuracy, provide an estimate of *feature importance,* can handle complex relationships, and are robust to noise and outliers (Breiman, 2001a; Breiman et al., 2017; Hastie et al., 2009).

Tree-based ML algorithms offer a flexible framework for modeling complex relationships and capturing nonlinear patterns in psychological phenomena, making them more suitable for the study of human behavior and cognition than linear models (Armey & Crowther, 2008; Breiman et al., 2017; Ehlers, 1995; Hastie et al., 2009). In addition, tree-based ML algorithms can effectively process high-dimensional data and automatically identify relevant features for prediction or classification tasks (Hastie et al., 2009). This capability is particularly advantageous in studies involving extensive psychological assessments, genetic data, or large-

scale surveys, where linear models may struggle to capture the complexity of multidimensional relationships (Abelson, 1995; Armey & Crowther, 2008; Chekroud et al., 2021; Dwyer et al., 2018; Jolly & Chang, 2019).

However, it is important to acknowledge the challenges associated with tree-based algorithms. DTs can be prone to overfitting, resulting in models that perform well on training data but fail to generalize to new data (Zhang et al., 2019). Additionally, tree-based methods cannot follow trends outside the training data range because they are based on mean estimation and therefore cannot extrapolate on new, unseen data. (Zhang et al., 2019).

Nevertheless, tree-based ML algorithms offer promising avenues to advance psychological research by capturing nonlinear relationships, handling high-dimensional data, and providing interpretability.

1.3 Linear Trees and Forests

While RFs have clear advantages and outperform linear models in most cases, for true linear relationships linear models yield more predictive power (Dwyer et al., 2018; Smith et al., 2013; Westreich & Greenland, 2013). Interestingly, the advantages and disadvantages of both methods are complementary, leading to the intuitive conclusion to combine the methods to compensate for the disadvantages of each approach (Zhang et al., 2019). By integrating linear regression into the nodes of tree-based ML models, they can capture both linear and nonlinear relationships, allowing for improved extrapolation while retaining the non-linear learning abilities of tree-based algorithms (Zhang et al., 2019).

Linear trees (LTs) use a tree-like graph or model that is constructed through an iterative process. Each node is split into child nodes based on specific rules, unless it becomes the final node or *leaf node*. To obtain predictions for new samples, a regression model is fitted to each leaf node, providing estimated values for the output variables (Hastie et al., 2009; James et al., 2013; Mitchell, 1997; Zhou, 2021). This approach can be extended to more sophisticated tree-based ML algorithms, such as RFs, making them *Linear Forests* (LFs).

The primary objective of LTs or LFs is to predict continuous variables based on input features. The combination of multiple linear regression and tree-based ML offers a compelling approach to overcome the limitations of each individual method (Yang et al., 2017; Zhang et al., 2019).

The idea of combining tree-based ML algorithms with other statistical models is not entirely new. After the Classification and Regression (CART) algorithm was introduced by Breiman et al. in 1984, it quickly gained popularity as a flexible and easily interpretable method (Breiman et al., 2017; Yang et al., 2017). In 1992, Quinlan developed the M5 model tree algorithm (M5') for classification tasks, which incorporated linear regression models at each node of the tree, improving interpretability and performance. This algorithm was further extended by Wang and Witten in 1997 and has been used by various researchers in different studies (Bonakdar & Etemad-Shahidi, 2011; Frank et al., 1998; Landwehr et al., 2005; Yang et al., 2017).

Following the M5', researchers continued to explore the integration of more sophisticated models within tree nodes. Notable algorithms include SUPPORT (Smoothed and Unsmoothed Piecewise-Polynomial Regression Trees) introduced by Chaudhuri in 1994, GUIDE (Generalized, Unbiased, Interaction Detection and Estimation) developed by Loh in 2002, SECRET by Dobra and Gehrke (2002), MART by Elish (2002), SMOTI by Malerbao et al. (2004), MAUVE by Vens and Blockeel (2006), BART by Chipman et al. (2010), and SERT by Chen and Hong (2010) as mentioned in Yang et al. (2017).

Several studies have demonstrated the effectiveness of model trees in various domains, for instance Landwehr et al. (2005) using M5' model trees. In the same year, Potts and Sammut employed linear models in tree nodes and observed favorable performance (Potts & Sammut, 2005). Bonakdar and Etemad-Shahidi (2011) successfully used classification model trees, and Yang et al. (2017) demonstrated the superiority of model trees over alternative models in their study. However, model trees are not yet widely employed in data analysis. Instead, methods such as RF and gradient boosting are preferred.

While most of the previous studies focused on classification problems, Zhang et al. (2019) used *regression enhanced forests*, which are comparable to the LFs introduced in this study, the difference being the use of *L1* or *lasso penalty* (Ranstam & Cook, 2018; Tibshirani, 1996) in their regression models, while this study employs *L2* or *ridge penalty* (Hoerl & Kennard, 1970; McDonald, 2009). Ridge penalty will be explained in Chapter 2.6. The study by Zhang et al. (2019) found that the regression enhanced forest significantly outperformed the RFs,

especially in extrapolation tasks. The study also emphasized the importance of hyperparameter tuning for regression enhanced forests (Zhang et al., 2019).

1.4 Thesis Statement

Exploring the potential of combining linear regression models and tree-based ML algorithms in the analysis of psychological data could benefit psychological research methods (Abelson, 1995; Breiman, 2001b; Chekroud et al., 2021; Dwyer et al., 2018; Rocca & Yarkoni, 2021; Westreich & Greenland, 2013). The purpose of this thesis is to investigate whether LTs and LFs outperform DTs and RFs when used in psychological research. A single psychological data set was used for all four estimators – DT, LT, RF, and LF – using relationship satisfaction as the target variable and 52 variables as input. For a full overview see tables 1-5. The performance of these estimators was evaluated using coefficients of determination (R^2), and the difference was tested for significance.

To formally test the hypotheses, the following null and alternative hypotheses were formulated:

H_o: Adding linear models to the (leaf) nodes of decision trees and random forests does not improve the performance of the algorithm when applied to our dataset.

H_1: Adding linear models to the (leaf) nodes of decision trees and random forests significantly improves the performance of the algorithm when applied to our dataset.

2 Methods

2.1 Overview

In this analysis, four supervised ML estimators - DT, LT, RF and LF - are tuned and trained on identical data. Their performance is subsequently evaluated using a *Student's T-test* (STT) (Johnston, 1970; Owen, 1965) or a *Welch's test* (WT) (Delacre et al., 2017; Derrick & White, 2016; Keselman et al., 2004) respectively if a significant difference in variances is found via the *F-test* (Edwards, 2005; Field et al., 2012; Lix et al., 1996). STT and WT are each performed with a *Nadeau-Bengio correction* (NBC) on the distribution of R^2 scores of each estimator (Nadeau & Bengio, 2003). A nested *cross-validation* (CV) technique is used to train and evaluate the ML models (Hastie et al., 2009; Hutter et al., 2015; Probst et al., 2019; Weerts et al., 2020). The "inner loop" of the CV (iCV) serves to tune the *hyperparameters* of the four ML models, while the "outer loop" (oCV) assesses the prediction quality. The hyperparameters of our ML models are the parameters that are set before the learning process begins (Hastie et al., 2009; Probst et al., 2019; Weerts et al., 2020). Unlike the parameters which are learned from the training data of the oCV, the hyperparameters are tuned in the iCV using a subset of the training data of the oCV. More information on the tuning process will be presented in chapter 2.4.2 and on the specific hyperparameters of the models in this analysis in chapters 2.5 – 2.8. The performance for tuning, training, and testing is evaluated using R^2 scores.

2.2 Data Description

The data used in this analysis were collected by Karremans et al. (2020) to examine the influence of mindfulness on relationship satisfaction. The sample consists of 989 Dutch couples, with data collected at three time points – before (pre), during, and after (post) the intervention - resulting in 2967 samples. Each sample consists of a participant (A) and their partner (B). As in the intervention group underwent a mindfulness training, while as in the control group received a relaxation instruction. Bs received no intervention.

The outcome variable to be predicted in this analysis is A's relationship satisfaction (A_RAS). 21 variables are excluded due to missing data, duplicate information, and lack of intelligibility (see Appendix, Table 1, Note). In addition, 950 samples with missing values are removed and

*k-Nearest-Neighbor (*KNN) *imputation* is used to fill in the 459 missing values of the variable "A_Income" with the training data of the oCV. The final dataset consists of 2017 samples and 53 variables. Significant changes in variable distributions due to the exclusion of 950 samples are found in the variables "A_ECR", "B_FFMQ", and "RDistress", as well as in A_Income due to KNN imputation using a STT for metric variables and binominal tests for non-metric variables. This may affect the interpretability of the results.

For a complete overview of all the variables, see Appendix: Table 1-5.

2.3 Analysis Pipeline

2.3.1 Data Preparation

Prior to data preparation, descriptive statistics such as *means, standard deviations* (std), or proportions are calculated and visualizations in the form of tables and graphs are created to ensure the variables follow a normal distribution.

Next, data coded as strings are transformed to numeric values and categorical data are dummy coded (Wolf & Cartwright, 1974).

21 variables and 950 samples are removed (Appendix: Table 1, Notes). The order of removing variables or samples with missing values first does not affect the amount of data excluded in this data set.

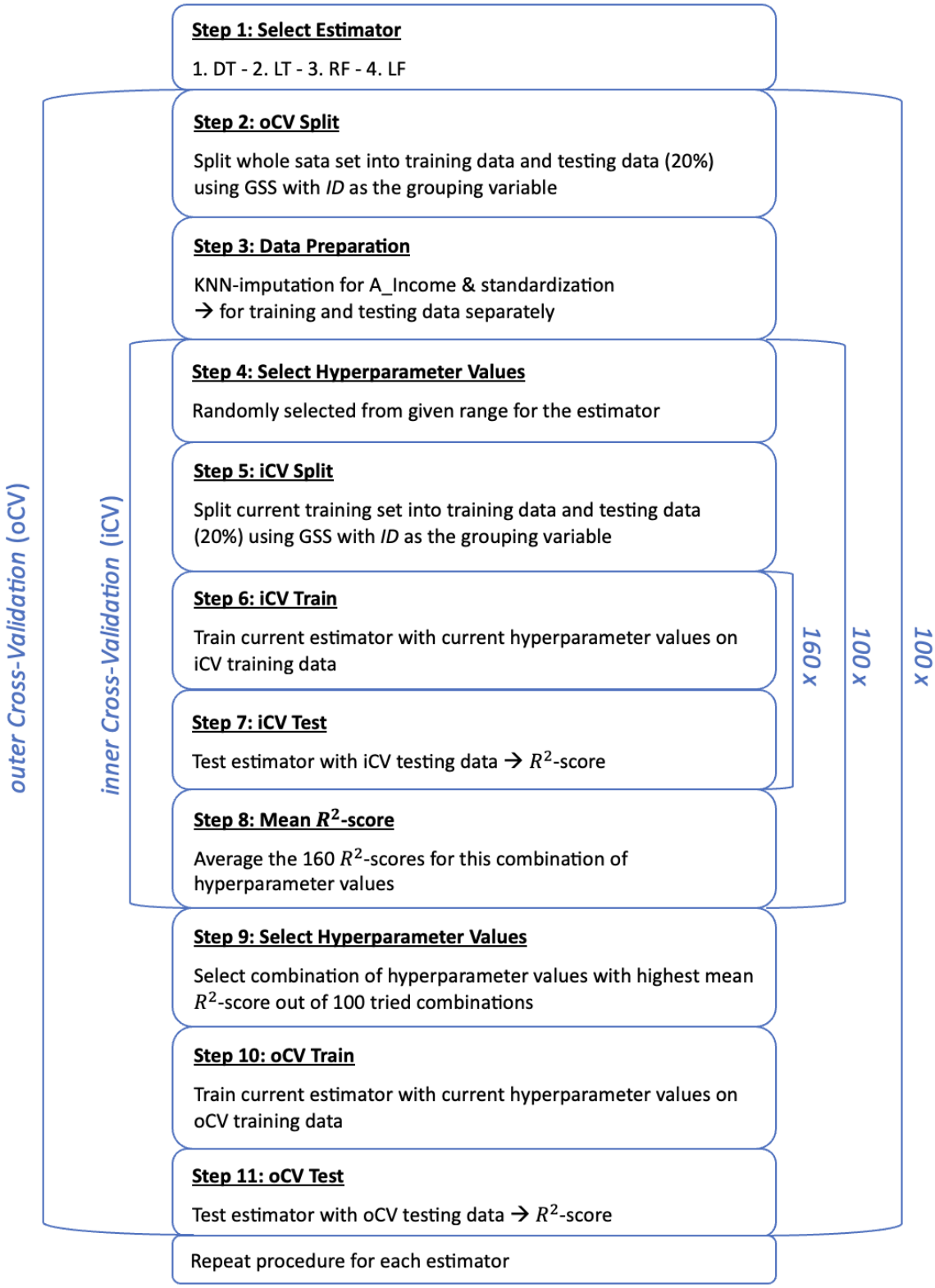
T-tests for metric data and binomial tests for categorical data are performed to control for significant changes in the mean of the distribution of each metric variable caused by the exclusion of 950 samples (Field et al., 2012; Johnston, 1970; Kaempf, 1995; Owen, 1965; Wagner-Menghin, 2005). Refer to Appendix: Tables 1-5 for comprehensive details on the data and the performed t-tests and binomial tests and to Appendix: Python code: *Script 1: Data Preparation* for the complete data preparation code.

2.3.2 Cross Validation

An overview of the CV procedure is shown in Figure 1.

Figure 1

Overview Cross-Validation



Note. Abbreviations used in figure 1: A_Income = participant income category, CV = cross-validation, GSS = GroupShuffleSplit, iCV = inner cross-validation loop, ID = identification variable, DT = decision tree, KNN = k-nearest-neighbor, LF = linear forest, LT = linear tree, oCV = outer cross-validation loop, R^2 = explained variance, RF = random forest

As preparation for the CV, all data except the target A_RAS and the grouping variable id (identification number of the couple) are designated as predictors (X). A_RAS is defined as the target (y) and id is used as the grouping variable (g) (Hastie et al., 2009; Little et al., 2017). The necessity of a grouping variable for this analysis will be explained in this chapter.

The models are defined as estimators 0-3 in the following order: DT, LT, RF, and LF (See Figure 1: "Step 1: Select Estimator"). The corresponding hyperparameters of each estimator are tuned in the iCV. The hyperparameters include: the minimum number of samples per leaf node (MSL) and the minimum number of samples per split node (MSS) for all estimators, the base estimator lambda for the ridge penalty in the linear regression (λ) (see chapter 2.6) (Hoerl & Kennard, 1970; McDonald, 2009; Melkumova & Shatskikh, 2017; Muthukrishnan & Rohini, 2016) for LT and LF, and the maximum proportion of features to consider for each split (max_features) for RF and LF. The base estimator lambda for the ridge penalty in the linear regression (λ) will be addressed in chapter 2.6.

The iCV and the oCV are implemented using *GroupShuffleSplit* (GSS) from Scikit-Learn with a test size of .2 out of 1, which is equivalent to 20% of the data, and the grouping variable id to ensure that samples with the same id are in the same set. The grouping is necessary to prevent information leakage since each couple provided up to three samples at different time points. Including samples from the same couple in both the training and testing data could lead to overfitting and overestimation of model performance (Hastie et al., 2009; Little et al., 2017).

The oCV enumerates the 100 oCV splits for each estimator, with the data being re-split into testing and training sets each time using GSS. A seed is set for the GSS of the oCV to ensure that each estimator has the same 100 train-test splits, thus ameliorating comparability. To scale the metric data to a mean of 0 and a std of 1, the Standard Scaler sklearn.StandardScaler is trained on the training data and applied on the training and testing data. Training the Standard Scaler on the whole data set, including the testing data, would lead to information leakage. KNN imputation is performed on the variable A_Income with a k-value of 5. Next, the RandomizedSearchCV (RSCV) method from Scitkit-Learn is used to tune the hyperparameters in the iCV by randomly sampling each hyperparameter value from a distribution over possible values. For each of the 100 value combinations tried per oCV loop the model gets trained and testing with that set of hyperparameter values 160 times. In our experience the prediction roughly 20,000 samples leads to a stable estimation of the generalization error. Incorporating the size of the data set and the train-test splits for the oCV and iCV, 160 iterations in the RSCV

leads to each hyperparameter value combination predicting the target value for roughly 20,000 samples.

The set of hyperparameter values resulting from the iCV is used to train the model on the training set. The trained model is then used to predict outcomes for the testing set, and the performance of the model is evaluated using R^2 scores of the true values and the predict values of the target variable.

The for loop then prints the descriptive statistics of the R^2 distribution for each estimator along with the elapsed computation time. Following the loop, a boxplot is generated to compare the R^2 distributions for each estimator.

For an overview of the complete CV procedure see Figure 1.

2.3.3 Comparison of the Estimators

Equal variances are required to use the STT (Delacre et al., 2017; Johnston, 1970; Owen, 1965). If an F-test renders a significant result, the assumption that the variances are equal is rejected (Edwards, 2005; Field et al., 2012; Lix et al., 1996). Consequently, three F-tests are performed to compare the variances of DT vs. LT, LT vs. RF, and RF vs. LF using the "f_test" method available in scipy.stats. If the variances are unequal, the WT can be used instead of the STT (Delacre et al., 2017; Derrick & White, 2016; Welch, 1947). Then three methods are defined; "corrected_std"/ "corr_std_error" to apply the NBC to the standard error, "compute_corrected_ttest" to compute a STT using "corrected_std" and "dep_two_sample_ttest" to compute a WT using "corr_std_error".

Three STTs and WTs respectively, each with NBC are performed for the same comparisons as mentioned above.

2.3.4 Exploratory Analysis

To explore the effect of the number of estimators (n_estimators), an additional nested CV generated new R^2 distributions for RF and LF with 10, 20, 50, 100, 200, and 300 estimators.

2.4 Cross Validation (CV)

2.4.1 Model Evaluation

The goal of this section is to explain the process of estimator evaluation, starting with the concept of CV, which is a widely applied method for training and testing ML models. The use of the error estimator R^2 and the scitkit-learn method GSS will also be discussed (Pedregosa, 2011). Additionally, the importance and placement of standardization and KNN imputation in this analysis will be addressed.

CV is a technique for evaluating ML models that involves partitioning the data into training sets, on which the model is trained, and testing sets, on which the model is evaluated (Hastie et al., 2009; Little et al., 2017; Westreich & Greenland, 2013). This process is repeated several times with different splits to ensure a more accurate estimation of the model parameters and it's performance, and to avoid any random error introduced by the specific partitioning of the data. The advantage of this approach is that the model is tested on different data than it was trained on, which increases the reliability of the evaluation (Hastie et al., 2009; James et al., 2013; Raschka & Mirjalili, 2019; Westreich & Greenland, 2013).

The measure R^2 is used to quantify the differences in model performance for each estimator. R^2 represents the proportion of the variance in the outcome that is explained by the model. The values of R^2 can range from negative infinity to 1, with a value of 1 indicating a perfect fit and a value of 0 indicating that the model performs just as well as the trivial predictor which is prediction by the average value. A negative R^2 value would indicate that the model's predictions are less accurate than the trivial predictor (Vanderplas, 2016).

To compute the R^2 score, the sum of the squared differences between the actual outcome values y_i and the predicted outcome values $\hat{y}_i$ is divided by the sum of the squared differences of y_i and the mean y_i (= $\bar{y}$) and subtracted from 1, as shown below:

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

(Field et al., 2012; Hastie et al., 2009)

This analysis uses the *scikit-learn* method GSS with 100 repetitions, to partition the data into testing in training sets. Unlike the widely used k-fold CV method, GSS randomizes the data for each split. This means that some samples may never be included in the testing set, while others may be included more than once.

To ensure comparability between the estimators, the random state of the GSS is fixed. This results in each estimator being trained and tested with the same 100 train-test splits.

Grouping the data by the variable "id" is also implemented to prevent samples of the same couples at different time slots from being in different sets, which could lead to overfitting and overestimation of the R^2 scores.

By training and testing each estimator 100 times, a distribution of oCV 100 R^2 scores per estimator is obtained.

Additional data preprocessing steps, namely standardization and KNN imputation, are performed within the CV loop after the train-test split. They are implemented after the split because they must be performed separately for the training and testing set to ensure consistency of scaling across folds and to avoid information leakage between the sets (Hastie et al., 2009).

Standardization of the continuous data is performed (*mean* = 0, std = 1), as is standard procedure for multiple linear regressions, to counteract multicollinearity (Daoud, 2017; Field et al., 2012).

KNN imputation is used to handle missing values in the "A_Income" variable. This method imputes missing values with the mean value of their k-nearest neighbors in the feature space (James et al., 2013; Kirk, 2017).

Alternatively, samples with missing values in A_Income could have been excluded or the feature itself removed, as was done for the other missing values. However, this would have meant excluding almost a quarter of the data or an interesting and possibly powerful feature. Another option would be *mean imputation*, which means replacing all missing values with the mean of the variable. However, mean imputation can artificially narrow standard errors and reduce variance. In addition, mean imputation does not take relationships between features into account and can lead to a biased analysis if the data is not *completely missing at random* (Donders et al., 2006; Mehrotra et al., 2017; Waljee et al., 2013). As KNN imputation also uses a mean, not the one of the complete data set but of the nearest neighbors, the disadvantages of mean imputation can also be of concern using the KNN imputation method. The number of

nearest neighbors used for the imputation, k, should be tuned via CV (Huang et al., 2017). However, to reduce the run time of this analysis, k was left at its default setting, k=5, instead. It is important to note that the application of KNN imputation affects the interpretability of the results, as the impact of income is not fully supported by actual data.

2.4.2 Hyperparameter Tuning

The performance of ML estimators is influenced by their hyperparameters. Finding good hyperparameter values is crucial for achieving optimal estimator performance. Hyperparameters are predetermined settings of ML estimators that are not learned from the data (Dwyer et al., 2018; Hastie et al., 2009; James et al., 2013). To ensure a fair comparison of the estimators, a set of hyperparameters and their possible ranges are defined separately for each estimator to create optimal conditions for each model.

In Section 2.4.1, CV was introduced as a technique for training and evaluating models. The same method is used to find good hyperparameter values. The approach involves trying different combinations of hyperparameter values on a subset of the training data and evaluating their performance on the rest of the training data - the *validation set* - with the goal of finding the hyperparameter values that yield the highest R^2 scores. This process is also known as hyperparameter tuning (Hutter et al., 2015; Probst et al., 2019; Raschka & Mirjalili, 2019; Weerts et al., 2020).

The process of hyperparameter tuning can again introduce the problem of information leakage between the data sets and overfitting, similar to standardization and KNN imputation. To avoid this problem, hyperparameter tuning should be performed on the training data only (Hastie et al., 2009; Probst et al., 2019; Weerts et al., 2020). The procedure is placed in the CV after the train-test split, the standardization, and the KNN imputation. Since it is a CV on the training set of another CV, it is called a *nested* CV. The iCV tunes the hyperparameters and is wrapped in the oCV, which trains the models and evaluates their performance. It is important that these two CVs are actually nested and not just separate, because if any of the testing data from the oCV were used to optimize the hyperparameters, the performance of the model could be overestimated (Cawley & Talbot, 2010; Hastie et al., 2009; Raschka & Mirjalili, 2019). The scikit-learn method RSCV is used to randomly sample hyperparameter values and evaluate the performance of the estimator with the sampled hyperparameter value set.

Compared to GridSearchCV, RSCV is computationally more efficient and often finds good hyperparameter values with fewer iterations (Hastie et al., 2009; Pedregosa, 2011; Raschka & Mirjalili, 2019). Within the RSCV, GSS is used to ensure correct grouping of the samples with the grouping variable “id”, and 160 splits are performed to validate each hyperparameter value set.

To obtain optimal results, it is essential to carefully select the most important hyperparameters to be tuned and to define an appropriate range for each parameter that includes the optimal values (Hastie et al., 2009; Hutter et al., 2015; Probst et al., 2019; Weerts et al., 2020). Unnecessarily wide ranges should be avoided (Hastie et al., 2009; Probst et al., 2019). For hyperparameters that take a continuous range of values, density functions are chosen to define the range, and for hyperparameters that take only discrete values, NumPy arrays are used (Hastie et al., 2009; Pedregosa, 2011).

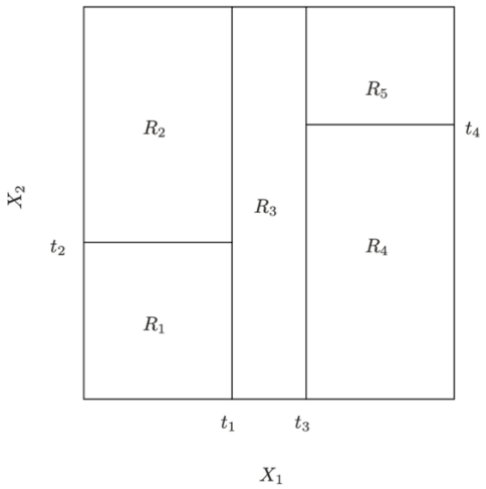
2.5 Estimator 1: Decision Tree (DT)

A DT is a non-parametric and intuitive supervised ML method designed to make predictions about the values of a target variable by learning simple decision rules based on a set of input features. DTs can be used for both classification and regression tasks. In this analysis, the four different estimators are used for a regression problem as the target A_RAS is a metric variable (Breiman et al., 2017; Hastie et al., 2009; James et al., 2021).

The basic idea behind DTs applied to regression problems is to recursively split the data into distinct, non-overlapping subsets (Figure 2.1) based on the values of the features until a stopping criterion is met. The result is an upside-down tree-like structure in which each node represents a test on a feature and each branch represents the possible outcomes (Figure 2.2). At each node, the mean of the target values for all the data in that node is calculated and serves as the prediction for all the data in that node. Thus, a DT can also be described as a stepwise approximation. The error between the mean prediction and the actual target values is also computed in each node. The first node is called the root node and contains all the training data, and the final nodes are called leaf nodes and represent the final prediction for the target value of their subset of data (Breiman et al., 2017; Hastie et al., 2009; James et al., 2021).

Figure 2.1

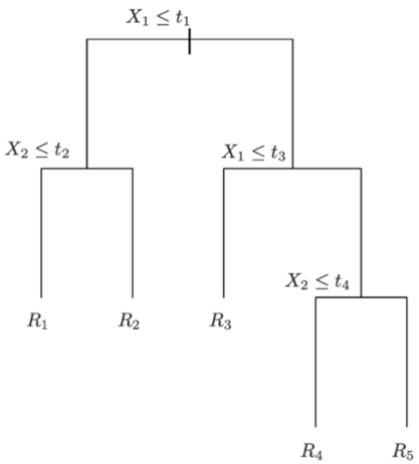
Distinct, Non-Overlapping Regions R_1- R_5 of a Decision Tree



Note. This is a two-dimensional example for illustrative purposes, with input features X_1 and X_2 and splitting values t_1- t_4 from *An Introduction to Statistical Learning* (2nd ed., p. 332), by G. James, D. Witten, T. Hastie, R. Tibshirani, 2021, Springer.

Figure 2.2

A Decision Tree corresponding to the partition in Figure 2.1



Note. This is a two-dimensional example for illustrative purposes, with input features X_1 and X_2 , leaf nodes R_1- R_5 and splitting values t_1- $t_4$from *An Introduction to Statistical Learning* (2nd ed., p. 332), by G. James, D. Witten, T. Hastie, R. Tibshirani, 2021, Springer.

There are several algorithms that can be used to create DTs, such as ID3, C4.5, C5.0, and CART. In this analysis, scikit-learn DecisionTreeRegressor() is selected, which uses the CART algorithm, as this is currently the most widely used DT algorithm. (Breiman et al., 2017; Hamze-Ziabari & Bakhshpoori, 2018; Zhang et al., 2019)

The algorithm for creating a DT involves selecting the best feature and its best value to partition the data at each node, which is the feature and the feature value whose partitioning most reduces the sum of the errors in its two child nodes. The *mean-squared error* (MSE) is used for error estimation in this analysis, as is common in DTs used for regression tasks (Das et al., 2004; Field et al., 2012; Wallach & Goffinet, 1989; Zhou Wang & Bovik, 2009). The data is then assigned to the two following nodes according to the split feature value, and the same process is repeated for each node. This algorithm is greedy, meaning that it does not plan ahead to create the best possible DT from the data, but instead considers only one split at a time and maximizes the error reduction at that split (Breiman et al., 2017; Hastie et al., 2009).

The performance of a DT is highly dependent on the hyperparameter values chosen for the model (Breiman, 2001b; Hastie et al., 2009; Hutter et al., 2015; Probst et al., 2019; Weerts et al., 2020). To limit the tree depth, the iCV tunes the MSL and MSS hyperparameters within a specified search range of 1 to 100 and 2 to 200, respectively. The MSL hyperparameter determines the minimum number of samples to be left in each leaf node, while the MSS hyperparameter controls the minimum number of samples for each interior node, thus for each split. These two hyperparameters are the stopping criteria for the DT to terminate the growing process. The remaining hyperparameters are kept at their default settings.

Once a DT is built, it can be used to make predictions on new data by traversing the tree from the root node to a leaf node based on the values of the input features. The mean estimation of the leaf node is the target prediction (James et al., 2021). The predictions of the model are then evaluated in the oCV by comparing the predictions to the actual target values of the testing sets using R^2 scores.

DTs have several advantages, including easy interpretability, applicability to both classification and regression problems, and the ability to model nonlinear relationships in data. They require

little data preparation and can handle both numeric and categorical input variables (Breiman et al., 2017; James et al., 2013).
However, DTs are neither smooth nor continuous, and unlike linear models they cannot follow linear trends outside the training data range. This means that if a sample is outside the training data range, the prediction for it is the mean estimate of the leaf node closest to its feature values. In addition, without an appropriate stopping criterion, DTs are prone to overfitting the data. Small variations in the data can lead to completely different DTs as DTs are unstable. Furthermore, the greediness of the CART algorithm can lead to suboptimal DTs (Breiman et al., 2017; Hastie et al., 2009).

While the accuracy of DTs alone cannot compete with some of the best supervised ML methods, several variations and extensions have been developed such as RFs and gradient boosted trees (Breiman, 2001a; Hastie et al., 2009; James et al., 2021).

2.6 Estimator 2: Linear Tree (LT)

LTs are a type of tree-based model that combines the advantages of linear models and DTs. The essential idea behind LTs is to use linear regression models to make predictions on the leaves of a DT, rather than simply using constant mean estimates to predict the target value. This approach may allow for smoothing the predictions of DTs to capture the linear relationships between features and target variables as well as the non-linear relationships (Zhang et al., 2019). Additionally, extrapolation is improved as the LT can follow trends outside its training data range.
The basic equation of linear regression, as presented in Section 1.1, can be translated into a n-dimensional hyperplane, with n being the number of input features, in an n+1-dimensional space, where each with dimension represents one variable, including the target (Field et al., 2012).
In the nodes of a LT, linear regression models are used to make predictions based on the input features of the data points assigned to that node.

To counter collinearity of the input features and to prevent the linear models in the leaf nodes from overfitting the data and therefore leading to poor generalization, ridge regression is used

in the leaf nodes. Ridge regression is a regularization technique that extends the traditional least squares method by adding a penalty term to the *residual sum of squares* (RSS) (Hastie et al., 2009; James et al., 2013).

Mathematically, ridge regression seeks to minimize the following function:

$$\text{RSS} + \lambda \sum_{j=1}^{p} \beta_j^2$$

(James et al., 2013)

Where *p* is the number of predictors, *j* the *j*-th predictor, β the coefficient or slope parameter and λ the ridge penalty parameter controlling the strength of the penalty (James et al., 2013). The added term $\lambda \sum_{j=1}^{p} \beta_j^2$ shrinks the slope parameters towards zero (James et al., 2013). This mitigates multicollinearity issues (Field et al., 2012; Hastie et al., 2009) and reduces the model's complexity, which can lead to improve generalization on the testing data or new data (Hastie et al., 2009; Hoerl & Kennard, 1970; James et al., 2013; McDonald, 2009; Melkumova & Shatskikh, 2017; Saleh et al., 2019; Vanderplas, 2016).

The ridge penalty parameter λ can be varied from zero to positive infinity, with zero indicating no effect on the regression hyperplane, and a numerically higher value indicating a higher reduction in the slope than a lower value for the ridge penalty parameter λ (Field et al., 2012; James et al., 2013). To determine the optimal value for the ridge penalty, λ is tuned in the iCV, just like the other hyperparameters, with a search range defined as a logarithmic uniform distribution between 10^{-3} and 10^{3} to ensure finer tuning at the lower values, e.g., selecting a similar number of values between 0.1 and 1 as between 1 and 10.

In addition, the hyperparameters used for the DT are also tuned for the LT, using the same search range as for the DT. The rest of the hyperparameters are all set to default.

The linear tree package by Marco Cerliani used in this study works by iteratively building a regular decision tree, fitting linear regression to the data points within each leaf node, and then repeating the process until a stopping criterion is met. Cerliani notes that the linear models are incorporated into all nodes. However, since the algorithm is based on the scikit-learn method RandomForest(), it remains unclear whether the linear models are solely located in the leaf nodes or also dispersed throughout the interior nodes. Note that to construct a single LT, the LinearForest() method is used with the number of estimators set to 1 and

bootstrapping disabled, as this has been shown to be more computationally efficient than the LinearTree() method in the same package.

2.7 Estimator 3: Random Forest (RF)

The third estimator being compared is the in comparison to the DT more sophisticated supervised machine learning method RF, which is implemented using the RandomForestRegressor() method from scikit-learn (Pedregosa, 2011).
The RF is a ML technique that combines many DTs to create a more accurate and robust model for classification or regression tasks. Therefore, it is expected to outperform the first two estimators, DT and LT (Breiman, 2001a; Breiman et al., 2017; Hastie et al., 2009; James et al., 2021). In this analysis the RF consists of 300 trees. This hyperparameter is not tuned, since more trees will most likely always result in a better prediction (Breiman, 2001a; Breiman et al., 2017; Hastie et al., 2009; James et al., 2021). I set the number of trees to 300 because I do not expect the increase in predictive power to be large enough to compensate for the increase in computational cost above an estimator number of 300. The many DTs of the RF are each trained on a different random subset of the training data, which is called bootstrapping, and at each node a random selection of the features is used to find the optimal split point. Bootstrapping and random feature selection both lead to a greater variety of trees. The outputs of all these individual DTs are then aggregated using the mean to generate the final prediction (Biau & Scornet, 2016; Breiman, 2001a; Hastie et al., 2009).
Overall, RFs are a highly flexible and effective ML technique that can be applied to a wide variety of data types and tasks. They are particularly useful when working with complex or noisy data, and often outperform other models with similar computational costs (Hastie et al., 2009).

The depths of the trees in the RF are limited by the hyperparameters MSS and MSL, and they are tuned in the same manner as for the single DT, except for the range of the MSL parameter being extended to up to 200. The higher range for the MSL hyperparameter is chosen because the scikit.learn method allows for it (Pedregosa, 2011). While the other methods have a limit for the MSL hyperparameter at 100, for the RF it is set at 200. Typically, the optimal depth of the individual DTs in the RF exceeds that of the single DT, resulting in each DT overfitting the

data (Breiman, 2001a; Hastie et al., 2009). This is commonly referred to as high variance error (Hastie et al., 2009).

However, since each DT in the RF is trained on a different subset of data and features, the high variance errors made by each DT are different. When the predictions of all the DTs are combined, the biases tend to cancel each other out, resulting in a more accurate overall prediction.

Another hyperparameter that is tuned in the iCV is the random selection of available features. To ensure that the DTs are different and produce different errors, in addition to bootstrapping, only a subset of features is used for each split. The size of the subset is determined by the "max_feature" hyperparameter, which represents a percentage of all the features available. The search space for this hyperparameter is defined as a uniform distribution ranging from 0.01 to 0.99. This hyperparameter also ensures that the random forest is not overly influenced by any single feature.

2.8 Estimator 4: Linear Forest (LF)

The present analysis uses ML algorithm for the LF which combines the learning ability and accuracy of RF with the predictive power of linear models (Zhang et al., 2019). This fourth estimator, called LF in this analysis, is a hybrid of the second estimator (LT) and the third estimator (RF) that integrates the best features of both methods. Given its unique combination of strengths, I expect LF to yield the highest R^2 values of all the estimators in this analysis (Frank et al., 1998; Landwehr et al., 2005; Yang et al., 2017; Zhang et al., 2019).

For the LF, the *linear-tree package* developed by Marco Cerliani is used, and the LF consists of 200 estimators. This number is 100 less than in the RF because LF is expected to achieve high performance with fewer estimators. The same hyperparameters were tuned as for the previous estimators, namely MSL, MSS, max_features and λ for ridge regression. The search ranges for these hyperparameters remain unchanged.

2.9 Comparison of the Estimators

2.9.1 Students' T-Test (STT), Welch's test (WT) and F-test

To test the differences in R^2 scores between DT and LT, LT and RF, and RF and LF for significance, the STT is selected as it tests the means for distributions for statistical significance (Field et al., 2012; Gravetter & Wallnau, 2017; Johnston, 1970). One requirement of the STT is that the variance of the two distributions to be compared is equal (Howell, 2013; Owen, 1965; Welch, 1947). The F-test is a widely used method to test the differences between the variances of two distributions for statistical significance (Field et al., 2012; Howell, 2013; Johnston, 1970).

Before conducting STTs, the F-test is calculated by dividing the variances of the two distributions by another (Delacre et al., 2017; Field et al., 2012; Howell, 2013; Wilkinson & Task Force on Statistical Inference, 1999). The result is compared to a critical value from the F distribution to obtain the p-value (Gravetter & Wallnau, 2017). If the p-value is below .05, the probability that the variances are equal is less than 5% and therefore considered significant in this analysis as the alpha level is set at 5%.

If the F-test does not show a significant difference between the variances of the R^2 distributions of two estimators to be compared, the STT is employed. The STT is a statistical test used to determine whether there is a significant difference between the means of two groups of normally distributed data with equal variances. It is based on the t-statistic, which is calculated by dividing the difference between the means of the two samples by the standard error of the difference (Field et al., 2012; Gravetter & Wallnau, 2017; Johnston, 1970; Owen, 1965).

If the F-test shows a significant difference between the variances of the R^2 distributions of two estimators to be compared, the STT cannot be employed as the requirement of equal variances is not met (Delacre et al., 2017; Field et al., 2012). In this case, the WT is performed. The WT works similar as the STT except that the standard error of the difference is not based on a pooled variance estimate. This makes the WT applicable to problems with different variances as well (Delacre et al., 2017; Derrick & White, 2016; Field et al., 2012; Keselman et al., 2004; Welch, 1947).

The resulting t-value derived from either the STT or WT is then compared to a critical value from a t-distribution to derive the p-value, which indicates the probability that the difference in means occurred by chance. A p-value below the 5% threshold indicates that the probability of the difference being due to chance is less than 5%, and thus the difference is considered statistically significant (Field et al., 2012).

2.9.2 Nadeau-Bengio Correction (NBC)

To ensure a valid comparison between ML estimators, the NBC was applied to each WT and STT to address the systematic underestimation of the variance introduced by the selection of testing and training sets. This correction is necessary because the testing and training sets are different subsamples of the original data and overlap at different iterations, violating the independence assumption required for proper significance testing. Using an uncorrected STT or WT in this scenario could result in a "significant" difference between the performance of two estimators where none exists, also known as a Type I error or alpha error. (Gravetter & Wallnau, 2017; Nadeau & Bengio, 2003)

To address this issue, Nadeau and Bengio proposed a correction that accounts for this dependence by adjusting the variance estimates (Nadeau & Bengio, 2003).

The NBC of the variances adjusts the systematically underestimated variance by taking into account the number of repetitions in the CV (Nadeau & Bengio, 2003).

2.10 Exploratory Analysis

2.10.1 Impact of Hyperparameters

One major goal is to gain insight into how stable an estimator performs across different hyperparameter values, which could be an advantage of this estimator. We used scatter plots and histograms to visualize the estimator performances across different hyperparameter values and find patterns within and between estimators. The transparency of individual dots in the scatterplots is set to a very low level (alpha = .01) to ensure that trends are visible, even when visualizing 40,000 pairs in a single plot (with 100 combinations tested per 100 oCV loops across four different estimators).

Histograms are employed to visualize how often each hyperparameter value is selected for the oCV.
This provides valuable insights into the strengths and limitations of the estimators, including their robustness and their need for extensive hyperparameter tuning, which affects their computational efficiency (Hastie et al., 2009; Hutter et al., 2015; Probst et al., 2019).

2.10.2 Impact of n-estimators on Random and Linear Forests

Tuning the hyperparameter n_estimators lacks sensibility, since more trees will likely always lead to better performance – so one could just right away set the highest value of the range for n_estimators (Breiman, 2001a; Hastie et al., 2009). The real question is: How many trees are needed to strike the perfect balance between performance and computational efficiency (Hutter et al., 2015; Probst et al., 2019; Weerts et al., 2020)?
In the case of RF, a value of 300 is often used as a default, since adding more trees beyond that point is unlikely to yield significant performance gains (Breiman et al., 2017; Hastie et al., 2009). For LF, the optimal value is likely to be lower, since the linear models in the leaf nodes are expected to compensate for a smaller number of trees.
The selected values for n_estimators in the main analysis are based on guesswork. To shed some light on the relationship between n_estimators and performance for RF and LF for future analyses, an additional analysis was performed using the same nested CV as the main analysis, but this time the estimators are RF and LF with n_estimators of 10, 20, 50, 100, 200, 300, resulting in a total of 12 models.
The goal of this analysis is to identify the point at which each RF and LF reached an optimal balance between performance and efficiency, and to see how that balance differed between the two. If the LF reached this point earlier than the RF, it could potentially be a more efficient option.

In summary, this thesis aims to evaluate and compare the performance of each estimator. Additionally, it explores the effect of the hyperparameters on the different estimators to gain valuable insight into the strengths and weaknesses of the different methods. The results of the analysis are presented in the following section.

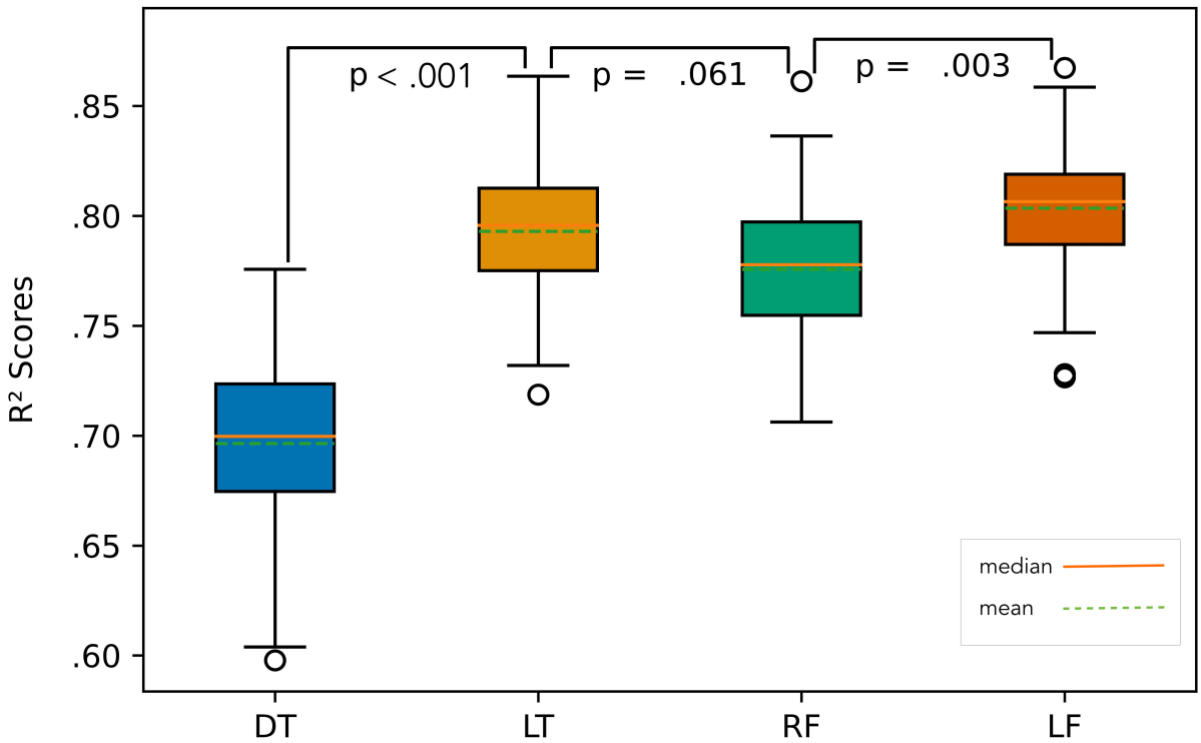
3 Results

3.1 Estimator Performances

In Figure 3, the distributions of the R^2 scores for each estimator are displayed.

Figure 3

Comparison of the R^2 Scores of the different Estimators



Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, p = p-value, R^2 = explained variance, RF = random forest

The performance of each estimator was evaluated based on their respective R^2 distributions obtained from the nested CV, considering mean, median, std, minimum, and maximum values. For all estimators, the mean and median were closely aligned. There was no clear pattern in the proximity of the minimum and maximum values to the mean.

Table 6 presents the mean, median, std, minimum and maximum R^2 values for each estimator, rounded to five decimal digits. Additionally, the last row displays the runtimes for the complete CV for each estimator in seconds, rounded to one decimal digit.

Table 6

Descriptive Statistics of the R^2 Distributions of the different Estimators

Statistic	DT	LT	RF	LF
Mean	.696	.793	.776	.804
Median	.700	.796	.778	.807
Std	.0409	.0279	.0308	.0267
Minimum	.598	.719	.706	.727
Maximum	.776	.864	.861	.867
Runtime	4039.6 s	6859.3 s	417030.5 s	363424.3 s

Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, R^2 = explained variance, s = seconds, Std = standard deviation, RF = random forest

3.2 Comparison of the Estimators

An F-test was performed to compare the estimators DT to LT, LT to LF, and RF to LF. The resulting p-values are shown in Table 7. If the F-test yielded a significant result, a WT was performed, otherwise a STT was used. The NBC was applied to both WT and STT. The p-values of the WT and STT are shown in Figure 3 and Table 7.

Table 7

P-values of F-Tests, WT or STT

Statistic	DT-LT	LT-RF	RF-LF
p-value F-Test	< .001	.833	.0793
p-value WT	< .001	-	-
p-value STT	-	.061	.003

Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, RF = random forest, STT = Student's T-Test, WT = Welch's test

3.3 Exploratory Analysis

3.3.1 Impact of Hyperparameters

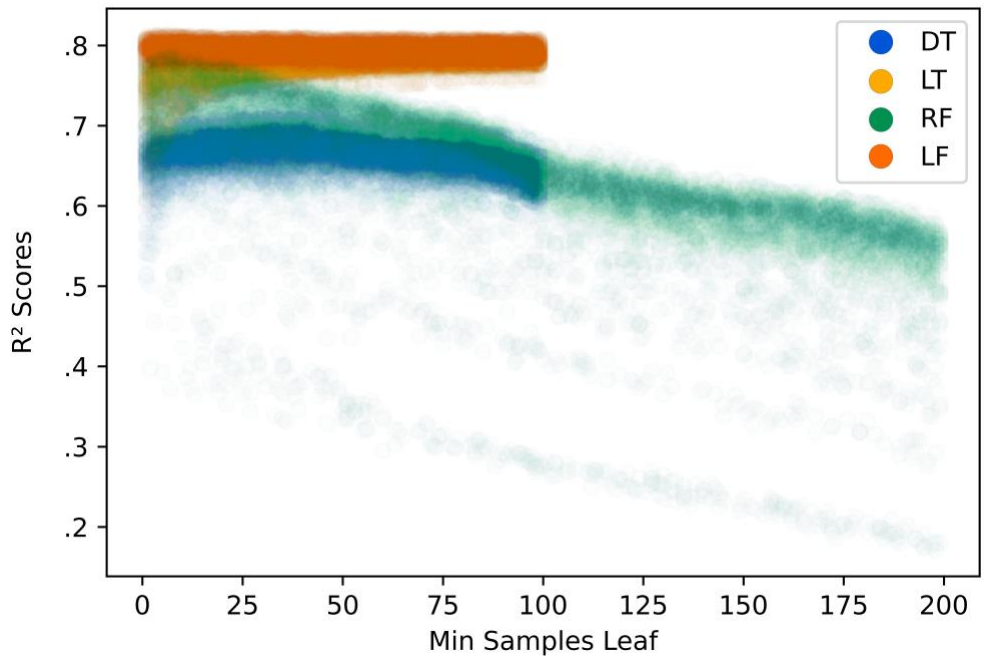
Figures 4-12 display the impact of the hyperparameters. For each tuned hyperparameter, a scatter plot and a histogram visualize its effect on the four different estimators.

The scatter plots (Figures 4, 6, 8 and 10) illustrate all values tried in the iCV with the corresponding R^2 scores as data points, for a total of 10,000 values per estimator – 100 values tried in the iCV for each of the 100 oCV iterations.

The 100 hyperparameter values that are selected as the best combination of hyperparameter values out of 100 in the iCV and used accordingly in the oCV, leading to the R^2 scores used for the main analysis, are depicted in histograms (Figures 5, 7, 9, and 11). The possible hyperparameter values are summarized into 10 bins of equal ranges.

Figure 4

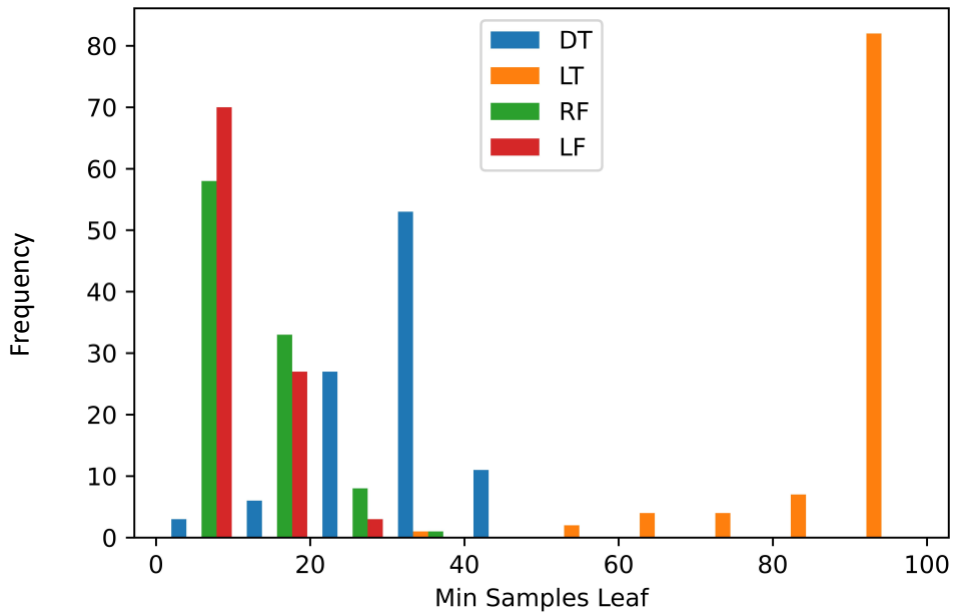
Impact of the Hyperparameter Min-Samples-Leaf on the Estimators Performance



Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, R^2 = explained variance, RF = random forest

Figure 5

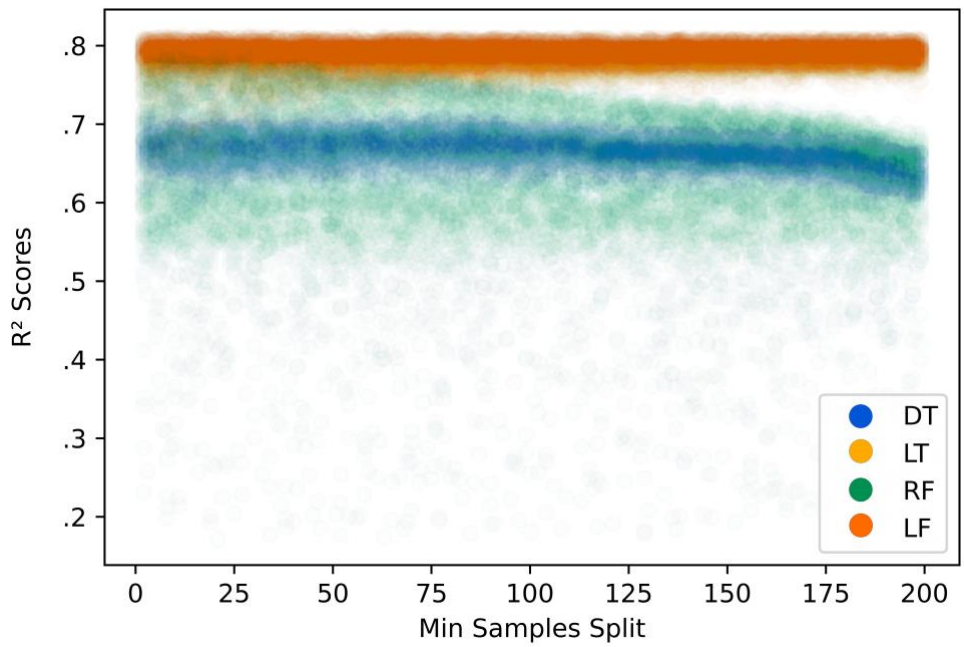
Best Values of the Hyperparameter Min_Samples_Leaf



Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, RF = random forest

Figure 6

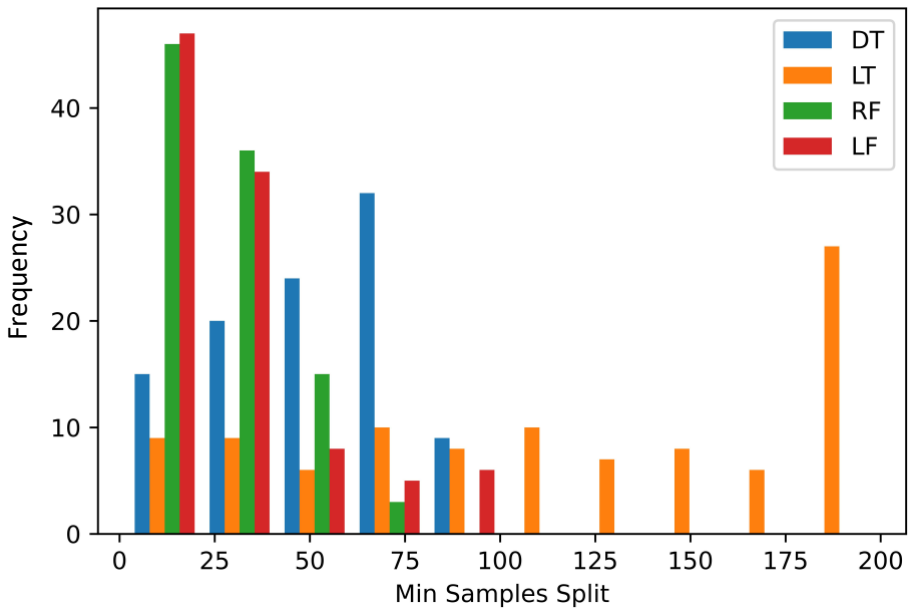
Impact of the Hyperparameter Min-Samples-Split on the Estimators Performance



Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, R^2 = explained variance, RF = random forest

Figure 7

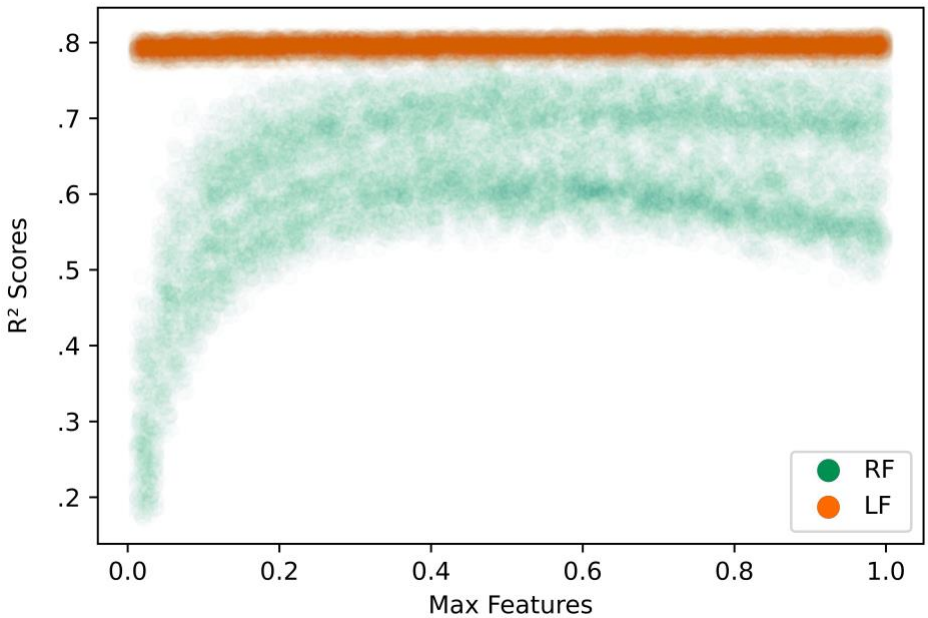
Best Values of the Hyperparameter Min_Samples_Split



Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, RF = random forest

Figure 8

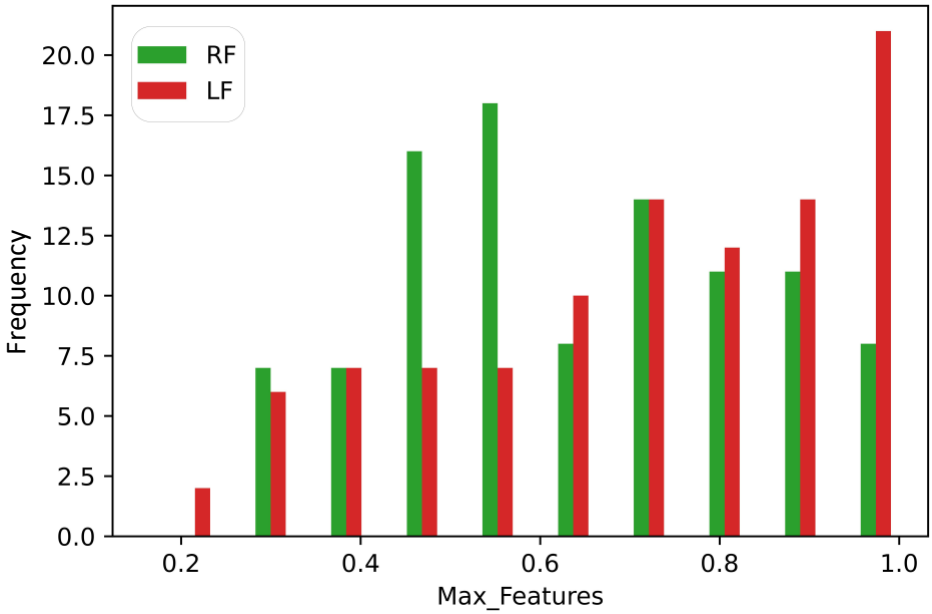
Impact of the Hyperparameter Max Features on the Estimators Performance



Note. Abbreviations used in this figure: LF = linear forest, Max Features = Maximum proportion of features to consider for each split, R^2 = explained variance, RF = random forest

Figure 9

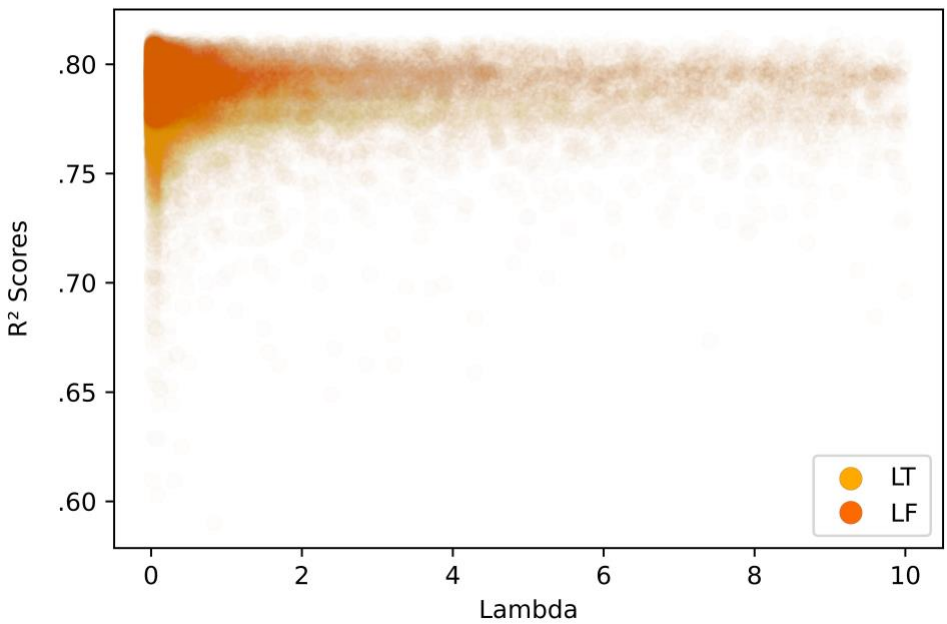
Best Values of the Hyperparameter Max_Features



Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, R^2 = explained variance, RF = random forest

Figure 10

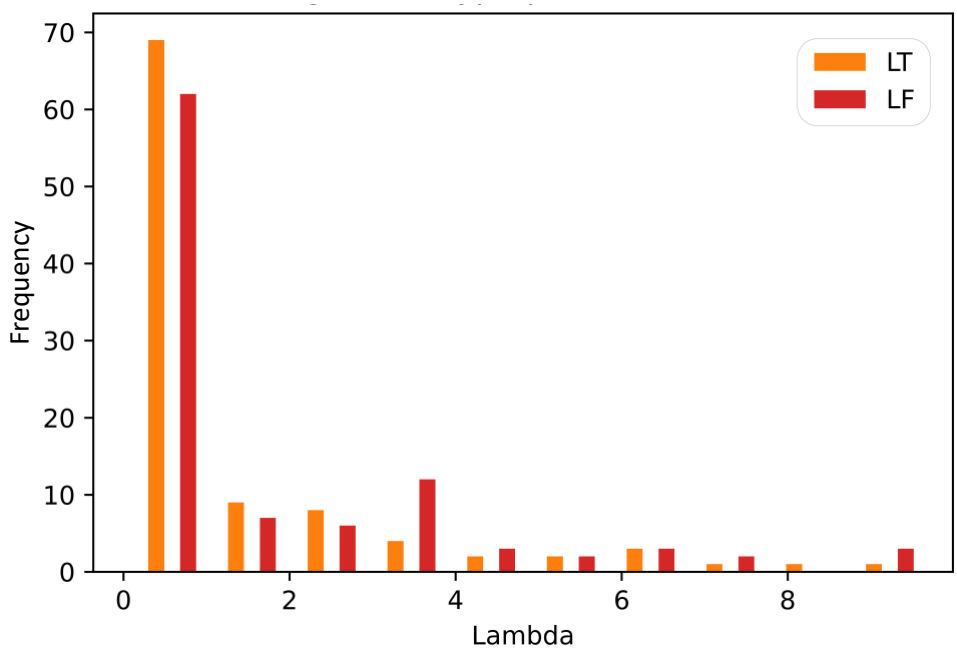
Impact of the Hyperparameter λ on the Estimators Performance



Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, R^2 = explained variance, RF = random forest

Figure 11

Best Values of the Hyperparameter λ



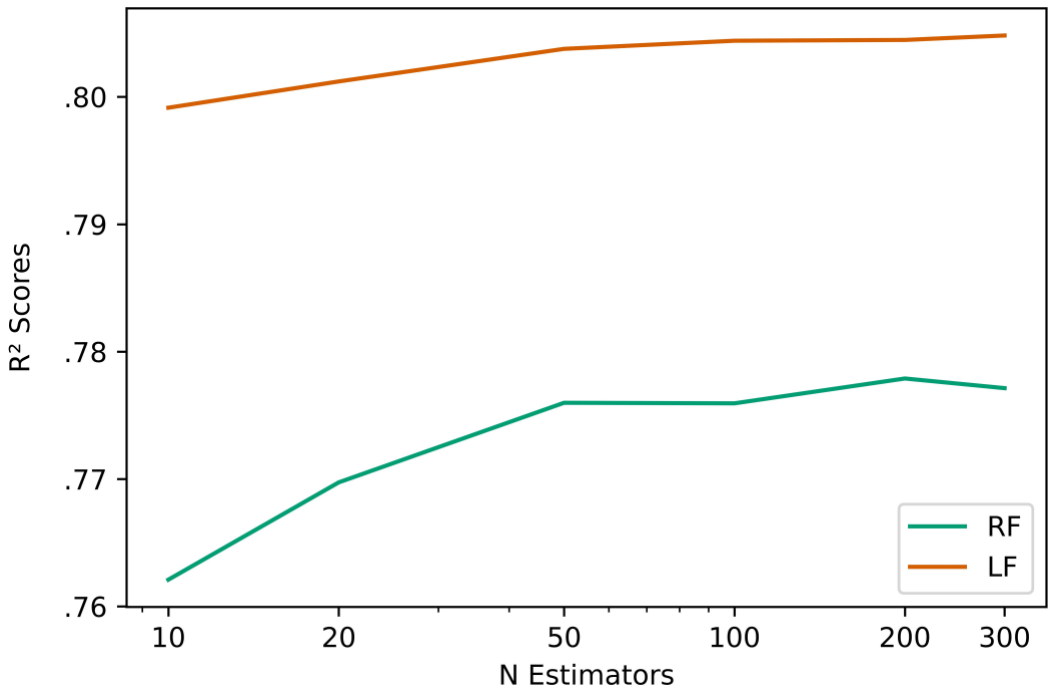
Note. Abbreviations used in this figure: DT = decision tree, LF = linear forest, LT = linear tree, R^2 = explained variance, RF = random forest

3.3.2 Impact of n-estimators on Random and Linear Forests

The results of the additional exploratory CV comparing the performance RF and LF with 10, 20, 50, 100, 200 and 300 estimators are depicted in Figure 12.

Figure 12

Impact of N_Estimators on RF and LF



Note. Abbreviations used in this figure: LF = linear forest, N Estimators = number of trees in a forest, R^2 = explained variance, RF = random forest

4 Discussion

This section focuses on analyzing the results provided by each of the four estimators, comparing their performance, examining the impact of hyperparameters, discussing the overall impact of the analysis, highlighting its limitations, and providing suggestions for further research.

4.1 Estimator Performances

The goal of this analysis is to compare the performance of four different estimators to determine if incorporating linear models in the nodes of tree-based ML models instead of mean prediction yields a significant improvement in predictive power. To ensure a fair comparison, all estimators were applied to the exact same data under equal conditions, including hyperparameter tuning.

From Figure 3 and Table 6 it is visible, that the mean and median of each estimator were closely aligned and their proximity was roughly equal to their maximum and minimum values, suggesting that the R^2 distributions were not skewed and that the mean was not influenced by outliers.

However, the values of the DT – excluding the std – are still visibly lower compared to the other estimators. Moreover, the DT displays a numerically higher variation compared to the other estimators, with a std of .04 and a comparatively very low minimum R^2 value of .60. The other estimators have stds of about .03 and the other minimum values do not fall below .70. This indicates comparatively low stability of the DT compared to the other estimators.

With a run time of 4039.6 seconds (equivalent to 1 hour, 7 minutes, and 19.6 seconds), the DT is the fastest method.

The LT estimator, which consists of a single DT augmented with linear models in the nodes, exhibits numerically higher performance than the DT and the RF with 300 trees. This suggests that the linear models integrated within the nodes compensate for having only one estimator. Moreover, the LT shows a relatively low std, indicating stability in performance.

With a total duration of 6859.3 seconds (equivalent to 1 hour, 54 minutes, and 19.3 seconds), the LT exhibits a longer execution time than the DT, although in a similar range, while

outperforming it. The achieved runtime is notably lower when compared to the forest estimators, which is remarkable considering the LT's ability to match the performances of said estimators.

With a total duration of 417030.5 seconds (equivalent to 4 days, 19 hours, 50 minutes and 30.5 seconds), the RF has the longest runtime.

The LF is the focus of this analysis and is intended to improve the widely used supervised ML method RF. The LF shows the numerically highest R^2 scores of all estimators combined with the lowest std, indicating its superiority and stability in predictive power.

The observed results of the LF are in line with the alternative hypothesis, or H_1, stating that linear models present an advantage over mean-based estimation in the nodes of tree-based ML models used for regression problems.

To assess the statistical significance of the differences in the R^2 distributions among the four estimators DT, LT, RF, and LF, an STT or WT is performed with NBC.

Specifically, DT is compared to LT, LT to RF, and RF to LF, as these comparisons are most relevant for answering the thesis statement whether including linear models in the nodes of DTs and RFs improves their performance on our dataset.

To determine whether the equal variances requirement of the STT is met, an F-test is performed on each set to be compared. A significant result indicates unequal variances, thus necessitating the use of the WT instead of the STT (Field et al., 2012; Gravetter & Wallnau, 2017).

As shown in Table 7, the F-test only yields a significant result for the comparison between DT and LT. This result is expected, considering the previously mentioned higher std of the DT. Consequently, a WT is employed for the comparison between DT and LT and the other estimators are compared via STT.

It is important to note that the non-significant F-test does not prove that the variances are equal, it only indicates that equality is not disproven at the significance level of .05 in our case. Assuming equal variances, even for the comparison of LT and RF with an F-test p-value as low as .08, can lead to an overestimation of the differences between the distributions.

The p-value for the DT-LT comparison using the less powerful WT is below .001, which is clearly significant considering our significance level at .05. Looking at the individual results of each

estimator, the significance of the WT is not surprising as the boxplots of the DT and LT only overlap at their whiskers (see Figure 3). The LT was expected to outperform the DT, but to have it's results in the same ballpark as the DT and serve mainly as the basis for the LF. However, while the DT is a very simple model by itself and typically only serves as the foundation of RF or gradient boosting, the LT shows comparable performance to the forest models RF and LF. The LT results overlap to a large part with the results of the RF and LF, numerically even surpassing the RF, while also showing similar stability as the forest models. The results of the LT greatly surpass the expectations.

The comparison between the RF and LF is crucial in assessing whether linear models offer a significant improvement over already sophisticated ML models. Hence, with the LT outperforming the DT significantly and the STT between the RF and LF resulting in a p-value of p-value of .003, our null hypothesis that the inclusion of linear models in the leaf nodes cannot improve the performance of DTs and RFs can be rejected. The comparative analysis underscores the significant differences in the R^2 distributions among the estimators, thus highlighting the effectiveness of incorporating linear models to improve predictive capabilities.

4.2 Impact of the Hyperparameters

The aim of this chapter is to evaluate the influence of the hyperparameters on the performance of the four different estimators by examining the R^2 scores in relation to the hyperparameter values selected through RSCV in the iCV of the main analysis.
Figures 5, 7, 9, and 11 use histograms to illustrate the selection patterns of hyperparameter value combinations in the iCV that were subsequently used in the oCV. The x-axis displays the hyperparameter values, while the y-axis represents the frequency of their use in the oCV. For the interpretation, it is important to note that each hyperparameter value is part of a hyperparameter value combination set, and the resulting R^2 score or choice to be used in the oCV may be due to the other hyperparameter values in the set.
In summary, each tuned hyperparameter is accompanied by a scatterplot and a histogram, allowing the identification of patterns in Figures 4-11. Additionally, the effect of the number of estimators is visualized by a line plot in Figure 12.

The MSL hyperparameter determines the minimum number of samples required in each leaf node. This affects the depth of each tree.

For the DT estimator, the MSL hyperparameter is especially important for finding the right balance between underfitting and overfitting. A large value leads to a very shallow tree with low complexity, while a small value can lead to a deeply grown tree that may overfit the data. The MSL scatter plot (Figure 4) shows a slightly curved line, indicating stable results with R^2 scores hovering around .65 and reaching a slight maximum for MSL values between 25 and 50. For MSL values above 50, the R^2 scores appear to drop slightly. The lower R^2 values could be influenced by other hyperparameters or the discrepancy between the testing and training data in that particular split.

The MSL histogram (Figure 5) shows a maximum of selected MSL values between 30 and 40 for the DT estimator, with more than half of the 100 selected MSL values in this range. No MSL values above 50 were selected.

The optimal MSL value of the DT is neither on the high nor low end of the search range, in order to avoid overfitting or underfitting the data. However, it is surprising that both very high and very low MSL values did not have a large effect on the resulting R^2 scores in the scatterplot. Regarding the high values, this could be attributed to the fact that requiring a minimum of 100 samples per leaf is not particularly stringent given the sample size of 2017, thereby resulting in less pronounced underfitting. As for the lower values, the lack of substantial drop in R^2 scores could be due to the moderation by the MSS hyperparameter. Even when the MSL hyperparameter is set to 1, if the MSS hyperparameter is set to 100, the impact on the tree depth and the subsequent overfitting will be less pronounced.

The DT and the LT could face similar challenges related to overfitting and underfitting. However, the LT is expected to be less susceptible to overfitting than the DT due to the inclusion of linear models, which would result in shallower trees and higher MSL values in the LT.

With MSL values falling below 50, the R^2 scores of the LT demonstrate a slight decline, indicating potential high variance errors for deeply grown LTs. This aligns well with the expected inclination of the single tree models such as DT and LT to overfitting with low MSL values. However, even the lowest R^2 scores appear to remain above .7, indicating that MSL only moderately affects LT performance.

The histogram shows a clear trend, with no MSL values below 30 being selected and approximately 80 of 100 MSL values falling within the range of 90-100. As approximately 80% of the MSL values are in the highest 10% of possible values, it is likely that MSL values above 100 are worth investigating for the LT. However, this was not possible as the LinearRegressor() method used in this analysis did not allow for the range of MSL values to exceed 100.

The RF is known to favor deeply grown trees as it can prevent overfitting by averaging over all 300 estimators (Breiman, 2001a; Hastie et al., 2009). Deeply grown trees counteract underfitting or bias error (Breiman, 2001a; Hastie et al., 2009). Therefore, the RF is expected to perform best with low MSL values (Breiman, 2001a; Hastie et al., 2009).

The RF shows a clear trend, with higher MSL values corresponding to lower R^2 scores. Across a range of MSL values from 1 to 200, the main trend line consistently decreases from approximately .8 to .6. Notably, parallel lines of RF R^2 scores appear below the main trend line, reaching as low as approximately .2. These lines are potentially influenced by other hyperparameters. It is striking that the lower lines appear to run almost perfectly parallel to the main trend line, suggesting that the effect of the MSL value on the performance of the RF may be direct and not moderated or mediated by other hyperparameter values.

The histogram illustrates the same trend of the RFs preference for low MSL values. No values above 40 were selected, and more than half of the selected values fall within the range of 0-10.

The LF is expected to show a similar but less extreme pattern than the RF, as it is expected to be more prone to overfitting and the linear models are expected to mitigate the high bias error resulting from shallow estimators.

In the scatter plot, the LF consistently exhibits a thick horizontal line just below .8 for all MSL values, indicating high stability. Surprisingly, the LF appears to perform well regardless of tree depth, as even forests with high MSL values yield excellent predictions. The robustness and insensitivity of the LF to MSL values suggests that tuning this hyperparameter may be unnecessary, giving it a computational advantage compared to the RF.

In the histogram, the LF shows a similar and even more pronounced trend compared to the RF. Approximately 70 of 100 selected values are in the range of 0-10, with no value exceeding 30. This implies that, although it is not visible in the scatter plot, the LF has a similar affinity for deeply grown trees as the RF.

Considering the R^2 scores from both iCV and oCV, it is unexpected and impressive that the LF consistently maintains high performance even with randomly selected hyperparameters.

The MSS hyperparameter is closely related to the MSL hyperparameter and also controls tree depth. While the MSL hyperparameter sets a minimum value for each leaf node, the MSS value sets the minimum for the last splits preceding the leaf nodes. Due to the nature of the relationship between MSL and MSS, the estimators are expected to show similar patterns for both hyperparameters, with values for MSS being approximately twice the values for MSL. Values from 2 to 200 are considered for all estimators. The expectations for each estimator are consistent with those previously stated for the MSL.

The MSS scatter plot (Figure 6) shows an almost horizontal line just below .7 for the DT. As the MSS values increase, there is a slight decline in the R^2 scores. This trend is only visible for MSS values over 100, indicating that shallow trees lead to a high bias error. Surprisingly, there is no discernible trend for deeply grown trees due to overfitting. This could be attributed to the influence of higher MSL values counterbalancing the impact of low MSS values and reducing or even eliminating their effect.

Below the DTs trend line in the MSS scatter plot, scattered R^2 scores are found ranging as low as .2. Since they do not exhibit a clear pattern, it is likely that they are caused by other hyperparameters and the train-test-split.

The MSS histogram (Figure 7) shows that all selected MSS values are in the range of 0-100, with most values in the range of 60-80. This corresponds well with the fact that most MSL values are in the range of 30-40.

The trend line for the LT is nearly identical to the one of the LF. Both estimators exhibit a perfect straight line at about .8, indicating stability and little effect of the MSS hyperparameter on their performance. The R^2 scores of the LT are slightly lower than those of the LF, indicating a small performance gap that is well in line with the overall performance of these estimators. Overall, both estimators are robust to changes in MSS values.

Figure 7 shows that selected MSS values for the LT are evenly distributed over the entire range of possible values, with a maximum at 180-200. The pattern is less extreme than that shown for the MSL hyperparameter, which could be due to the MSL hyperparameter balancing the MSS hyperparameter.

The RF shows a transposed funnel pattern, with a slightly wider range of on average slightly higher R^2 scores for lower MSS values, indicating a preference for deeper trees accompanied by higher variability. However, the slope of the transposed funnel is not steep, suggesting that the effect of MSS values on the RF's performance is relatively small. Notably, the RF's R^2 scores show a large variation, indicating sensitivity to other hyperparameters or the train-test split, or both.

This suggests that lower MSS values are favored by the RF, but only lead to good performance in combination with the other hyperparameters being set at values preferred by the RF. This leads to a great variation at the lower MSS values.

The histogram shows the preference for lower MSS values, with the maximum at 0-20, a clear decrease in frequency with increasing MSS values, and no MSS values above 100. The same pattern is displayed for the LF estimator, with no values selected above 80.

The max_features hyperparameter was compared for the RF and the LF. It determines the fractional number of features to consider at each split point. Manipulating this hyperparameter can lead to lower performance of individual trees because of bias errors, since they are unable to use the optimal feature at each split. However, it can enhance the variation between trees in the forest, thereby improving the overall performance of the forest as the variation errors are reduced through the aggregation of the trees (Biau & Scornet, 2016; Breiman, 2001a; Hastie et al., 2009; Probst et al., 2019). Values for max_features range from .01 to .99.

Regarding the RF, the scatter plot of max_features (Figure 8) reveals that the R^2 scores generally remain low, dropping as low as .2 for the lowest max_features values. As the max_features values increase from 0 to .2, there is a steep rise in R^2 scores. However, for max_features values higher than .2, the R^2 scores still vary widely between .45 and .75. These observations suggest that the RF performs poorly when only a small fraction of features is considered. Literature suggests however, that small max_feature values lead to the best RF results (Breiman, 2001a). The low R^2 scores for the RF with low max_feature values in this analysis indicate that the RF is dependent on other hyperparameter values for its optimal performance and thereby support the claim that the RF is not stable and requires hyperparameter tuning (Biau & Scornet, 2016; Breiman, 2001a; Hutter et al., 2015; Probst et al., 2019; Weerts et al., 2020).

The max_features histogram (Figure 9) shows the maximum frequency of selected max_features values of the RF in the range of .5 to .6. The remaining values are broadly distributed across the other ranges of hyperparameter values, except for the range from 0-.2 where no values are selected. This means that including approximately half of the features at each split gives the RF an advantage in predicting the data used in this analysis.

For the LF, the scatter plot shows a single perfect line with very little variation, indicating that the max_features hyperparameter has no noticeable effect on its performance. The R^2 scores remain consistently around .8, regardless of the max_features values. This indicates that the LF is not affected by this hyperparameter and maintains stable performance.

The frequency of the hyperparameter values used in the oCV is widely distributed in the histogram, with the maximum in a range from .9 to 1. Very low hyperparameter values below .1 were not selected, as this would likely result in suboptimal trees. This leads to the conclusion that adding more diversity to the trees by not including all features at each split may be less useful in LFs than in RFs. A possible explanation would be that the predictive power of the LF is already very high and therefore difficult to improve.

The λ hyperparameter is used for the LT and LF estimators as they both contain linear models to which the ridge penalty can be added. The range of λ values explored is from 0.01 to 10, including a finer grid at the lower values due to logarithmic steps.

For the LT, the λ scatter plot (Figure 10) reveals that lower λ values between 0 and 0.5 occasionally result in lower R^2 scores. However, above λ values of approximately 0.5-1, there is no visible difference in R^2 scores. The effect of λ on the LT performance seems to be limited. 70% of the λ values selected for the oCV, as presented in the histogram (Figure 11), are less than 1, indicating that a strong penalty to outliers does not improve the model performance. However, larger λ values of all ranges from 0-10 are included in the oCV.

LF's scatter plot shows that the R^2 scores are consistently distributed roughly around .8, regardless of the λ value. The ridge penalty does not appear to have an influence on the LF's performance. On another data set with higher correlation between the predictors, the influence of the λ value could have a stronger influence on the performance of the LF (Hoerl & Kennard, 1970; McDonald, 2009). 60% of the λ values selected for the main oCV are less than 1, which imposes only a small penalty on outliers. However, lambdas of 9-10 are also included.

Both LT and LF show stability regarding the λ hyperparameter, with a preference for λ values below 1.

Observation of the scatter plots and histograms (Figures 4-11) reveals that the performance of the DT and the RF is strongly influenced by hyperparameter tuning, and possibly also by the testing and training sets. This is evidenced by the large variation in their R^2 scores. Extensive tuning is crucial for these estimators to achieve optimal performance. In contrast, the LT and particularly the LF show a remarkable degree of stability in their R^2 scores and are less sensitive to hyperparameters and the train-test split. The LF consistently delivers strong performance, suggesting that hyperparameter tuning may not be necessary.

These findings regarding the influence of hyperparameters on the performance of the different estimators suggest that the LF's advantage in stability may also translate into efficiency if hyperparameter tuning can be omitted. By using a random set of hyperparameter values or default values, the LF has the potential not only to outperform the RF, but also to improve computational efficiency. The robustness of the LF concerning its hyperparameter values is interesting, as it contradicts the conclusions drawn by Zhang et al. (2019) in their study on RERF with lasso penalties.

The main analysis employed a value of 300 for the n_estimators hyperparameter, which denotes the number of trees in the forest, for the RF and 200 for the LF. Increasing the number of trees in the RF is generally beneficial in terms of performance, but the goal is to strike a balance between computational efficiency and high R^2 scores (Breiman, 2001a). At this point in time, the RF does not have an established default value for the number of trees at which point performance gains beyond this value are expected to be marginal, leading to diminishing returns. Most papers and analyses suggest numbers ranging from 100 to 500 with some even going as low as 10 or up to 2000 (Biau & Scornet, 2016; Genuer et al., 2010; Hastie et al., 2009; Perner, 2012). A value roughly in the middle at 300 was selected for this analysis. The LF, on the other hand, does not have any guidelines for the number of estimators yet, so 200 trees were chosen as a reasonable value, with the expectation that the LF would require fewer trees than the RF to achieve the desired balance.

A line graph (Figure 12) compares the results of repeated training and testing of the RF and the LF, similar to the main analysis, but varying the number of estimators to 10, 20, 50, 100, 200, and 300.

For the RF, the R^2 scores exhibit a steep increase until reaching 50 estimators, with the highest R^2 score at 200 estimators. Beyond 50 estimators, the trend is less apparent, with R^2 scores roughly equal for n_estimators of 50 and 100, and slightly lower for 300. At this point, there is no plausible explanation for the descending R^2 score with higher numbers of estimators, suggesting that this pattern is likely due to chance, with "chance" being a result of different hyperparameter values explored in the iCV, different testing and training sets in the oCV and the randomness within the RF caused by bootstrapping and the features selected for the splits.

For the LF, a modest increasing trend is observed between 10 and 50 estimators, with the line flattening after 50, showing minimal further increase. From this figure, it appears that 50 estimators may be an appropriate choice for the LF. However, even with the lowest number of 10 estimators, the LF's R^2 score is already approaching .8.

These results suggest that the number of estimators has a greater impact on the RF than on the LF, as the RF's R^2 scores vary more.

It is noticeable that the difference in R^2 scores between 10 and 50 estimators is smaller for LF compared to RF. This suggests that a smaller number of trees in the LF can still produce high quality results compared to the RF.

While the results do not provide a definitive answer, they suggest the possibility of working with a smaller number of trees in the LF while obtaining comparable performance to the RF.

4.3 Impact: The Potential of Linear Forests in Psychological Research

In the field of psychology, linear models have long been used to investigate relationships between variables, despite the fact that many of these associations are inherently nonlinear (Abelson, 1995; Armey & Crowther, 2008; Dwyer et al., 2018; Judd et al., 2011). As a result, numerous nonlinear effects and relationships have been overlooked, leading to erroneous conclusions and the dismissal of significant associations (Abelson, 1995; Armey & Crowther, 2008; Dwyer et al., 2018; Ehlers, 1995; Jolly & Chang, 2019; Judd et al., 2011; Westreich & Greenland, 2013). By harnessing the power of nonlinear modeling, valuable insights can be gleaned from these complex relationships (Chekroud et al., 2021; Dwyer et al., 2018; Westreich & Greenland, 2013). In this chapter, we examine the impact of LFs, a novel

approach that combines the strengths of linear and nonlinear tree-based models, enhancing our understanding of psychological phenomena.

Traditional linear models fail to capture the nuanced relationships present in psychological data because they assume linearity between variables. This limitation underscores the need to explore alternative methods that can account for nonlinear effects. By using tree-based ML algorithms, nonlinear relationships between variables can be effectively modeled, allowing researchers to uncover valuable insights that would otherwise go unnoticed.

While tree-based models excel at capturing nonlinear effects, they often lack the predictive power of linear models to model linear relationships and have difficulty to extrapolate to unseen data beyond the scope of the training data set. However, by incorporating linear models into the nodes of DTs, LTs or LFs can be constructed. This hybrid approach allows the integration of both linear and nonlinear factors, harnessing the strengths of each modeling technique. By leveraging the interpretability of linear models and the flexibility of tree-based ML models, researchers can improve the predictive ability and generalizability of their models.

To assess the potential of the LF in psychological research, I used a comprehensive dataset that includes commonly used variables in psychology, including the widely employed Big Five personality scores and demographic measures such as age. The analysis compares the performance of different estimators and shows that the LF consistently outperforms the other estimators, indicating its significant potential as a valuable addition to psychological research methods.

A potential drawback associated with employing complex models such as the LF is the high computational cost incurred during training and evaluation. However, the LF outperforms the RF at a lower run time. Unlike the RF, the LF requires only a small number of estimators and exhibits remarkable stability in terms of hyperparameter tuning. Consequently, default hyperparameter settings are likely to be sufficient, making the LF significantly more computationally efficient than the RF.

By adopting the hybrid approach of LFs, researchers can effectively model a wide range of complex relationships that reflect the intricacies inherent in psychological data. By

transcending the limitations of linear models, psychological research can unlock new insights and unveil previously unrecognized patterns and associations.

4.4 Limitations

The purpose of this section is to discuss the limitations of the analysis. First, I address the use of only one data set, then the statistical comparisons and then the implementation of the linear models within the tree-based models. Finally, the lack of a direct comparison with multiple linear regression and the need for additional analyses to gain insight into the computational efficiency of the LF estimators are acknowledged.

The performance of DT and RF with mean estimation and the hybrid approach of LT and LF showed significant results in this analysis, however the comparison takes place using only one specific data set. Depending on the kind of data, the size of the data set and the nature of the relations between the feature, different methods have advantages and disadvantages (Field et al., 2012; Gravetter & Wallnau, 2017; Hastie et al., 2009). LT and LF outperforming DT and RF on this data set does not mean that the same would be the case for other data sets. Therefore, a key limitation of this analysis is the use of only one data set to compare the estimators (Brandt et al., 2014; Jayaratne & Levy, 1979; King, 1995; Wilkinson & Task Force on Statistical Inference, 1999).

The use of the STT instead of the to compare the RF and the LF is questionable. Since the F-test results were marginally non-significant, the equality of the variances between the R^2 distributions of the two estimators has not been disproven at a significance level of .05. But it also has not been proven and a WT would have been the safer choice (Edwards, 2005; Field et al., 2012; Gravetter & Wallnau, 2017; Howell, 2013).

The extent to which linear models are implemented within the LT and the LF estimators is unclear at this point. This is a technical issue of the LinearForest() method used in this analysis. While the author of the linear-trees package, Marco Cerliani, indicates that linear models are used in all nodes, including the interior nodes, the package is based on the existing random forest module from scikit-learn, which suggests that a traditional random forest regression is run first, and the linear models are added only in the leaf nodes. This would mean that the

linear models do not influence the splits within the LTs. Clarifying this aspect would improve the understanding of this analysis.

While the LF estimator is introduced as an alternative to the primary model used in psychological research, multiple linear regression, a direct comparison between these two models was not made in this analysis. However, it is worth noting that the original study by Karremanns et al. (2020), which collected the data used in this analysis, employed multiple linear regression as its main analytical approach. Due to differences in analytical methods, including the treatment of missing values and the use of CV techniques, a meaningful comparison between the present analysis and the original analysis by Karremanns et al. (2020) is not possible.

The claim that LF is more computationally efficient than RF is based on the assumption that the LF requires fewer estimators and less hyperparameter tuning than the RF. However, the analysis did not reveal clear trends regarding the number of estimators, and the conclusions drawn about hyperparameter tuning are inconsistent with the study by Zhang et al. (2019). While the LF showed superior performance with a small number of estimators, further investigation is needed to prove the higher computational efficiency of the LF compared to the RF.

4.5 Suggestions for Future Research

In light of the limitations identified in the previous chapter, this section presents a series of recommendations for future research. These suggestions aim to bridge knowledge gaps, replicate findings, compare alternative models, explore different data sets, and refine the analysis of the LF estimator.

To validate the significance of the results obtained in the RF vs. LF comparison, the comparison should be replicated with different data (Brandt et al., 2014; Jayaratne & Levy, 1979; King, 1995). This would help to determine whether the observed significant differences are due to chance or whether they are robust and reliable findings. It would also provide insight into the generalizability of the LF by applying the estimator to data sets with different characteristics. In addition to assessing the versatility of the LF as a predictive model, this approach could also

help to find the underlying pattern that determines whether the LF works well on a prediction problem.

To gain a comprehensive understanding of the advantages of LF models in psychological research, it is crucial to compare them directly with the widely used multiple linear regression model on many different data sets. This study only examined the benefits of adding linear models to tree-based models on one data set. By comparing LF models with multiple linear regression, researchers could gain insight into the advantages of modeling nonlinear relationships, which is a crucial part of the benefits of the hybrid approach through the LF estimator.

To better understand how hyperparameter tuning affects the performance of the LF, researchers should conduct an analysis that includes training the LF with minimal or no hyperparameter tuning. This investigation would shed light on the inherent capabilities of the LF and provide insight into its performance without extensive optimization. Comparing the results to those obtained with hyperparameter tuning would offer valuable information about the tradeoff between model performance and computational cost when using the LF.
To optimize the tradeoff between model performance and computational cost when using LFs, additional research should be devoted to identifying the number of estimators that hit the point of diminishing returns. Investigating the optimal balance would improve the practical utility of LF models.

This chapter has outlined several suggestions for further research that can extend upon the findings and address the limitations identified in the previous chapter. Replication studies on different data sets, comparisons with multiple linear regression, analyses with minimal hyperparameter tuning for the LF, and further investigation of the number of estimators would contribute to a more comprehensive understanding of the performance and applicability of LFs in psychological research.

4.6 Conclusion

The purpose of this analysis was to explore the potential of LFs as a novel approach to potentially improve our understanding of psychological phenomena by improving our predictions and analyses. The evaluation of LF models with psychological research data shows a remarkable performance and stability of the LF estimator with respect to hyperparameter values. It is even possible that the LF models offer higher computational efficiency due to their stability. In the main analysis, the LF showed superior performance compared to the RF, indicating its significant potential as a valuable addition to psychological research methods.

Traditional linear models in psychology overlook nonlinear relationships, leading to erroneous conclusions and the dismissal of significant associations (Armey & Crowther, 2008; Austin et al., 1998; Jolly & Chang, 2019; Smith et al., 2013). By harnessing the versatility of nonlinear modeling through tree-based supervised ML models combined with the power of linear models, LFs allow researchers to uncover valuable insights that would otherwise go unnoticed. By seamlessly combining the strengths of linear and nonlinear modeling, researchers can effectively capture the complex relationships inherent in psychological data (Dwyer et al., 2018; Frank et al., 1998; Jolly & Chang, 2019; Landwehr et al., 2005; Potts & Sammut, 2005; Yang et al., 2017; Zhang et al., 2019). This hybrid approach provides a comprehensive understanding of the intricate phenomena under investigation, overcoming the limitations of traditional linear models.

5 References

Abelson, R. P. (1995). *Statistics as principled argument*. L. Erlbaum Associates.

Armey, M. F., & Crowther, J. H. (2008). A comparison of linear versus non-linear models of aversive self-awareness, dissociation, and non-suicidal self-injury among young adults. *Journal of Consulting and Clinical Psychology*, *76*(1), 9–14. https://doi.org/10.1037/0022-006X.76.1.9

Austin, E. J., Willock, J., Deary, I. J., Gibson, G. J., Dent, J. B., Edwards-Jones, G., Morgan, O., Grieve, R., & Sutherland, A. (1998). Empirical models of farmer behaviour using psychological, social and economic variables. Part I: Linear modelling. *Agricultural Systems*, *58*(2), 203–224. https://doi.org/10.1016/S0308-521X(98)00066-3

Barkley, R. A. (2011). *Barkley deficits in executive functioning scale (BDEFS)*. Guilford Press.

Biau, G., & Scornet, E. (2016). A random forest guided tour. *TEST*, *25*(2), 197–227. https://doi.org/10.1007/s11749-016-0481-7

Bohlmeijer, E., ten Klooster, P. M., Fledderus, M., Veehof, M., & Baer, R. (2011). Psychometric Properties of the Five Facet Mindfulness Questionnaire in Depressed Adults and Development of a Short Form. *Assessment*, *18*(3), 308–320. https://doi.org/10.1177/1073191111408231

Bonakdar, L., & Etemad-Shahidi, A. (2011). Predicting wave run-up on rubble-mound structures using M5 model tree. *Ocean Engineering*, *38*(1), 111–118. https://doi.org/10.1016/j.oceaneng.2010.09.015

Brandt, M. J., IJzerman, H., Dijksterhuis, A., Farach, F. J., Geller, J., Giner-Sorolla, R., Grange, J. A., Perugini, M., Spies, J. R., & Van 'T Veer, A. (2014). The Replication Recipe: What makes for a convincing replication? *Journal of Experimental Social Psychology*, *50*, 217–224. https://doi.org/10.1016/j.jesp.2013.10.005

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

Breiman, L. (2001b). Statistical Modeling: The Two Cultures (with comments and a rejoinder by the author). *Statistical Science*, *16*(3). https://doi.org/10.1214/ss/1009213726

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (2017). *Classification And Regression Trees* (1st ed.). Routledge. https://doi.org/10.1201/9781315139470

Cawley, G. C., & Talbot, N. L. C. (2010). On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. *Journal of Machine Learning Research*, *11*, 2079–

2107.
Chekroud, A. M., Bondar, J., Delgadillo, J., Doherty, G., Wasil, A., Fokkema, M., Cohen, Z., Belgrave, D., DeRubeis, R., Iniesta, R., Dwyer, D., & Choi, K. (2021). The promise of machine learning in predicting treatment outcomes in psychiatry. *World Psychiatry*, *20*(2), 154–170. https://doi.org/10.1002/wps.20882
Daoud, J. I. (2017). Multicollinearity and Regression Analysis. *Journal of Physics: Conference Series*, *949*, 012009. https://doi.org/10.1088/1742-6596/949/1/012009
Das, K., Jiang, J., & Rao, J. N. K. (2004). Mean squared error of empirical predictor. *The Annals of Statistics*, *32*(2). https://doi.org/10.1214/009053604000000201
Delacre, M., Lakens, D., & Leys, C. (2017). Why Psychologists Should by Default Use Welch's t-test Instead of Student's t-test. *International Review of Social Psychology*, *30*(1), 92–101. https://doi.org/10.5334/irsp.82
Derrick, B., & White, P. (2016). Why Welch's test is Type I error robust. *The Quantitative Methods for Psychology*, *12*(1), 30–38. https://doi.org/10.20982/tqmp.12.1.p030
Donders, A. R. T., Van Der Heijden, G. J. M. G., Stijnen, T., & Moons, K. G. M. (2006). Review: A gentle introduction to imputation of missing values. *Journal of Clinical Epidemiology*, *59*(10), 1087–1091. https://doi.org/10.1016/j.jclinepi.2006.01.014
Dwyer, D. B., Falkai, P., & Koutsouleris, N. (2018). Machine Learning Approaches for Clinical Psychology and Psychiatry. *Annual Review of Clinical Psychology*, *14*(1), 91–118. https://doi.org/10.1146/annurev-clinpsy-032816-045037
Edwards, A. W. F. (2005). R.A. Fischer, statistical methods for research workers, first edition (1925). In *Landmark Writings in Western Mathematics 1640-1940* (pp. 856–870). Elsevier. https://doi.org/10.1016/B978-044450871-3/50148-0
Ehlers, C. L. (1995). Chaos and Complexity: Can It Help Us to Understand Mood and Behavior? *Archives of General Psychiatry*, *52*(11), 960. https://doi.org/10.1001/archpsyc.1995.03950230074010
Field, A. P., Miles, J., & Field, Z. (2012). *Discovering statistics using R*. Sage.
Fraley, R. C., Waller, N. G., & Brennan, K. A. (2000). An item response theory analysis of self-report measures of adult attachment. *Journal of Personality and Social Psychology*, *78*(2), 350–365. https://doi.org/10.1037/0022-3514.78.2.350
Frank, E., Wang, Y., Inglis, S., Holmes, G., & Witten, I. H. (1998). Using Model Trees for Classification. *Machine Learning*, *32*, 63–76.
Genuer, R., Poggi, J.-M., & Tuleau-Malot, C. (2010). Variable selection using random forests. *Pattern Recognition Letters*, *31*(14), 2225–2236.

https://doi.org/10.1016/j.patrec.2010.03.014
Gravetter, F. J., & Wallnau, L. B. (2017). *Statistics for the behavioral sciences*. Cengage Learning.
Hamze-Ziabari, S. M., & Bakhshpoori, T. (2018). Improving the prediction of ground motion parameters based on an efficient bagging ensemble model of M5′ and CART algorithms. *Applied Soft Computing*, *68*, 147–161. https://doi.org/10.1016/j.asoc.2018.03.052
Harrison, M. (2019). *Machine learning pocket reference: Working with structured data in Python* (First edition). O'Reilly.
Hastie, T., Tibshirani, R., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed). Springer.
Hendrick, S. S. (1988). A Generic Measure of Relationship Satisfaction. *Journal of Marriage and the Family*, *50*(1), 93. https://doi.org/10.2307/352430
Hoerl, A. E., & Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, *12*(1), 55–67. https://doi.org/10.1080/00401706.1970.10488634
Howell, D. C. (2013). *Statistical methods for psychology* (8th ed). Wadsworth Cengage Learning.
Huang, J., Keung, J. W., Sarro, F., Li, Y.-F., Yu, Y. T., Chan, W. K., & Sun, H. (2017). Cross-validation based K nearest neighbor imputation for software quality datasets: An empirical study. *Journal of Systems and Software*, *132*, 226–252. https://doi.org/10.1016/j.jss.2017.07.012
Hutter, F., Lücke, J., & Schmidt-Thieme, L. (2015). Beyond Manual Tuning of Hyperparameters. *KI - Künstliche Intelligenz*, *29*(4), 329–337. https://doi.org/10.1007/s13218-015-0381-0
James, G., Witten, D., Hastie, T., & Tibshirani, R. (Eds.). (2013). *An introduction to statistical learning: With applications in R*. Springer.
James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An Introduction to Statistical Learning: With Applications in R*. Springer US. https://doi.org/10.1007/978-1-0716-1418-1
Jayaratne, S., & Levy, R. L. (1979). *Empirical clinical practice*. Columbia University.
Johnston, L. W. (1970). Student's *t* -Test. *Journal of Quality Technology*, *2*(4), 243–245. https://doi.org/10.1080/00224065.1970.11980443
Jolly, E., & Chang, L. J. (2019). The Flatland Fallacy: Moving Beyond Low–Dimensional Thinking. *Topics in Cognitive Science*, *11*(2), 433–454. https://doi.org/10.1111/tops.12404
Judd, C. M., McClelland, G. H., & Ryan, C. S. (2011). *Data Analysis* (0 ed.). Routledge.

https://doi.org/10.4324/9780203892053
Kaempf, U. (1995). The binomial test: A simple tool to identify process problems. *IEEE Transactions on Semiconductor Manufacturing*, *8*(2), 160–166. https://doi.org/10.1109/66.382280
Kappen, G., Karremans, J. C., Burk, W. J., & Buyukcan-Tetik, A. (2018). On the Association Between Mindfulness and Romantic Relationship Satisfaction: The Role of Partner Acceptance. *Mindfulness*, *9*(5), 1543–1556. https://doi.org/10.1007/s12671-018-0902-7
Karremans, J. C., Kappen, G., Schellekens, M., & Schoebi, D. (2020). Comparing the effects of a mindfulness versus relaxation intervention on romantic relationship wellbeing. *Scientific Reports*, *10*(1), 21696. https://doi.org/10.1038/s41598-020-78919-6
Keselman, H. J., Othman, A. R., Wilcox, R. R., & Fradette, K. (2004). The New and Improved Two-Sample *t* Test. *Psychological Science*, *15*(1), 47–51. https://doi.org/10.1111/j.0963-7214.2004.01501008.x
King, G. (1995). Replication, Replication. *PS: Political Science & Politics*, *28*(3), 444–452. https://doi.org/10.2307/420301
Kirk, M. (2017). *Thoughtful machine learning with Python: A test-driven approach* (First edition). O'Reilly.
Landwehr, N., Hall, M., & Frank, E. (2005). Logistic Model Trees. *Machine Learning*, *59*, 161–205.
Little, M. A., Varoquaux, G., Saeb, S., Lonini, L., Jayaraman, A., Mohr, D. C., & Kording, K. P. (2017). Using and understanding cross-validation strategies. Perspectives on Saeb et al. *GigaScience*, *6*(5). https://doi.org/10.1093/gigascience/gix020
Lix, L. M., Keselman, J. C., & Keselman, H. J. (1996). Consequences of Assumption Violations Revisited: A Quantitative Review of Alternatives to the One-Way Analysis of Variance *F* Test. *Review of Educational Research*, *66*(4), 579–619. https://doi.org/10.3102/00346543066004579
McDonald, G. C. (2009). Ridge regression. *Wiley Interdisciplinary Reviews: Computational Statistics*, *1*(1), 93–100. https://doi.org/10.1002/wics.14
Mehrotra, D. V., Liu, F., & Permutt, T. (2017). Missing data in clinical trials: Control-based mean imputation and sensitivity analysis. *Pharmaceutical Statistics*, *16*(5), 378–392. https://doi.org/10.1002/pst.1817
Melkumova, L. E., & Shatskikh, S. Ya. (2017). Comparing Ridge and LASSO estimators for data analysis. *Procedia Engineering*, *201*, 746–755. https://doi.org/10.1016/j.proeng.2017.09.615

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.
Muthukrishnan, R., & Rohini, R. (2016). LASSO: A feature selection technique in predictive modeling for machine learning. *2016 IEEE International Conference on Advances in Computer Applications (ICACA)*, 18–20. https://doi.org/10.1109/ICACA.2016.7887916
Nadeau, C., & Bengio, Y. (2003). Inference for the Generalization Error. *Machine Learning*, *52*(3), 239–281. https://doi.org/10.1023/A:1024068626366
Owen, D. B. (1965). The Power of Student's *t* -test. *Journal of the American Statistical Association*, *60*(309), 320–333. https://doi.org/10.1080/01621459.1965.10480794
Pedregosa, F. (2011). Scikit-learn: Machine Learning in Python. *JMLR*, *12*, 2825–2830.
Perner, P. (Ed.). (2012). *Machine Learning and Data Mining in Pattern Recognition: 8th International Conference, MLDM 2012, Berlin, Germany, July 13-20, 2012. Proceedings* (Vol. 7376). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-31537-4
Potts, D., & Sammut, C. (2005). Incremental Learning of Linear Model Trees. *Machine Learning*, *61*(1–3), 5–48. https://doi.org/10.1007/s10994-005-1121-8
Probst, P., Wright, M. N., & Boulesteix, A. (2019). Hyperparameters and tuning strategies for random forest. *WIREs Data Mining and Knowledge Discovery*, *9*(3). https://doi.org/10.1002/widm.1301
Rammstedt, B., & John, O. P. (2007). Measuring personality in one minute or less: A 10-item short version of the Big Five Inventory in English and German. *Journal of Research in Personality*, *41*(1), 203–212. https://doi.org/10.1016/j.jrp.2006.02.001
Ranstam, J., & Cook, J. A. (2018). LASSO regression. *British Journal of Surgery*, *105*(10), 1348–1348. https://doi.org/10.1002/bjs.10895
Raschka, S., & Mirjalili, V. (2019). *Python machine learning: Machine learning and deep learning with Python, scikit-learn, and TensorFlow 2* (Third edition). Packt.
Rocca, R., & Yarkoni, T. (2021). Putting Psychology to the Test: Rethinking Model Evaluation Through Benchmarking and Prediction. *Advances in Methods and Practices in Psychological Science*, *4*(3), 251524592110268. https://doi.org/10.1177/25152459211026864
Russell, S. J., Norvig, P., & Davis, E. (2010). *Artificial intelligence: A modern approach* (3rd ed). Prentice Hall.
Saleh, A. K. M. E., Arashi, M., & Kibria, B. M. G. (2019). *Theory of ridge regression estimators with applications*. John Wiley & Sons.
Smith, P. F., Ganesh, S., & Liu, P. (2013). A comparison of random forest regression and multiple linear regression for prediction in neuroscience. *Journal of Neuroscience Methods*, *220*(1), 85–91. https://doi.org/10.1016/j.jneumeth.2013.08.024

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (Second edition). The MIT Press.

Tabachnick, B. G., Fidell, L. S., & Ullman, J. B. (2019). *Using multivariate statistics* (Seventh edition). Pearson.

Tibshirani, R. (1996). Regression Shrinkage and Selection Via the Lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, *58*(1), 267–288. https://doi.org/10.1111/j.2517-6161.1996.tb02080.x

Vanderplas, J. T. (2016). *Python data science handbook: Essential tools for working with data* (First edition). O'Reilly Media, Inc.

Wagner-Menghin, M. M. (2005). Binomial Test. In B. S. Everitt & D. C. Howell (Eds.), *Encyclopedia of Statistics in Behavioral Science* (p. bsa743). John Wiley & Sons, Ltd. https://doi.org/10.1002/0470013192.bsa743

Waljee, A. K., Mukherjee, A., Singal, A. G., Zhang, Y., Warren, J., Balis, U., Marrero, J., Zhu, J., & Higgins, P. D. (2013). Comparison of imputation methods for missing laboratory data in medicine. *BMJ Open*, *3*(8), e002847. https://doi.org/10.1136/bmjopen-2013-002847

Wallach, D., & Goffinet, B. (1989). Mean squared error of prediction as a criterion for evaluating and comparing system models. *Ecological Modelling*, *44*(3–4), 299–306. https://doi.org/10.1016/0304-3800(89)90035-5

Weerts, H. J. P., Mueller, A. C., & Vanschoren, J. (2020). *Importance of Tuning Hyperparameters of Machine Learning Algorithms*. https://doi.org/10.48550/ARXIV.2007.07588

Welch, B. L. (1947). THE GENERALIZATION OF 'STUDENT'S' PROBLEM WHEN SEVERAL DIFFERENT POPULATION VARLANCES ARE INVOLVED. *Biometrika*, *34*(1–2), 28–35. https://doi.org/10.1093/biomet/34.1-2.28

Westreich, D., & Greenland, S. (2013). The Table 2 Fallacy: Presenting and Interpreting Confounder and Modifier Coefficients. *American Journal of Epidemiology*, *177*(4), 292–298. https://doi.org/10.1093/aje/kws412

Wilkinson, L. & Task Force on Statistical Inference. (1999). Statistical methods in psychology journals: Guidelines and explanations. *American Psychologist*, *54*(8), 594–604. https://doi.org/10.1037/0003-066X.54.8.594

Wolf, G., & Cartwright, B. (1974). Rules for coding dummy variables in multiple regression. *Psychological Bulletin*, *81*(3), 173–179. https://doi.org/10.1037/h0035848

Yang, L., Liu, S., Tsoka, S., & Papageorgiou, L. G. (2017). A regression tree approach using mathematical programming. *Expert Systems with Applications*, *78*, 347–357.

https://doi.org/10.1016/j.eswa.2017.02.013
Zhang, H., Nettleton, D., & Zhu, Z. (2019). *Regression-Enhanced Random Forests* (arXiv:1904.10416). arXiv. http://arxiv.org/abs/1904.10416
Zhou Wang, & Bovik, A. C. (2009). Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures. *IEEE Signal Processing Magazine*, *26*(1), 98–117. https://doi.org/10.1109/MSP.2008.930649
Zhou, Z.-H. (2021). *Machine learning* (S. Liu, Trans.). Springer.

6 Index of Figures and Tables

6.1 Index of Figures

6.2 Index of Tables

7 List of Abbreviations

Abbreviation	Meaning
A	Study participants
A_Income	Income categories of study participant
A_RAS	Measured relationship satisfaction of study participant; target variable
B	Romantic partners of study participants
CART	Classification and regression algorithm was introduced by Breiman et al. in 1984
CV	Cross-validation
DT	Decision tree
g	Grouping variable
GSS	GroupShuffleSplit
iCV	Inner cross-validation loop
id	Identification number of A-B couple
KNN	k-nearest neighbor
LF	Linear forest
LT	Linear tree
M5’	M5 model tree algorithm developed by Quinlan (1992) incorporating linear regression models
max_features	Maximum proportion of features to be considered in a split
ML	Machine learning
MSE	Mean squared error
MSL	Minimum samples per leaf node
MSS	Minimum samples per split
NBC	Nadeau-Bengio correction
n_estimators	Number of estimators; number of trees in the forest
oCV	Outer cross-validation loop
R^2	Coefficient of determination
RERF	Regression Enhanced Random Forest

RF	Random forest
RSCV	RandomizedSearchCV
RSS	Residual sum of squares
std	Standard deviation
STT	Student's T-test
WT	Welch's test
x	Independent variables, features
y	Target variable
λ	Lambda, penalty parameter for ridge regression

8 Appendix

8.1 German Abstract

Diese Arbeit untersucht das Potential der Kombination von *tree-based Machine-Learning* Methoden mit linearen Modellen in psychologischer Forschung. Gängige statistische Modelle wie die lineare Regression können die komplexen Zusammenhänge psychologischer Daten nicht ausreichend darstellen. Im Gegensatz dazu können *Random Forests* zwar nichtlineare komplexe Zusammenhänge modellieren, jedoch nicht außerhalb des Trainingsbereichs extrapolieren oder lineare Beziehungen effizient abbilden.

Linear Trees und *Linear Forests* kombinieren die Stärken beider Ansätze, indem lineare Modelle in *Decision Trees* und *Random Forests* integriert werden. Vier Methoden – *Decision Tree, Linear Tree, Random Forest* und *Linear Forest* – sagen unabhängig voneinander die Beziehungszufriedenheit von Proband*innen des selben Datensets vorher. Die Beurteilung erfolgt anhand des Bestimmtheitsmaßes, welches aus einer verschachtelten *Kreuzvalidierung* berechnet wird. Die *Kreuzvalidierung* dient neben dem *Trainieren* und Evaluieren der Methoden auch dem Anpassen der *Hyperparameter*. Zum Vergleich der Leistung werden *klassische T-Tests* und *Welch-Tests* mit *Nadeau-Bengio* Korrektur herangezogen. Zudem wird der Einfluss der Hyperparameter visuell dargestellt.

Die Ergebnisse zeigen, dass die Integration von linearen Modellen zu einer signifikanten Verbesserung der Vorhersagekraft und zu Stabilität bezüglich der *Hyperparameter* und *Train-Tests*-Aufteilungen führt. Der *Linear Forest* übertrifft die anderen Methoden hinsichtlich Genauigkeit und Stabilität unter den vier Methoden. Dabei übersteigt die Berechnungseffizienz des *Linear Forest* die des *Random Forest*.

Diese Analyse verdeutlich das Potenzial der Integration von linearen Modellen in *tree-based Machine-Learning* Methoden zur Weiterentwicklung des Methodenfundus der psychologischen Forschung.

8.2 Table 1 - 5

Table 3

Descriptive Statistics of Metric Data

Name	Measurement	Participants (A)					Partners (B)				p-value
		pre		post			pre		post		
		mean	std	mean	std	p-value	mean	std	mean	std	
Accept	*Partner Acceptance*. 5-item *Partner Acceptance Scale* (Kappen et al., 2018) ranging from 1 (low) to 5 (high)	5.22	1.05	5.23	1.05	.741	5.02	1.04	5.03	1.04	.739
Affect	*General Affect.* Single item from 1 (not at all well) to 7 (very well)	5.26	1.21	5.27	1.21	.775	5.26	1.25	5.27	1.25	.782
Age	*Age*. Measured in years	48.75	14.00	49.40	14.10	.109	49.76	13.85	50.50	13.90	.065
B5_Ag	*Agreeableness*. 10-item version of the *Big-5 questionnaire* (Rammstedt & John, 2007) from 1 (low) to 7 (high)	4.27	1.09	4.29	1.11	.528	4.30	1.14	4.32	1.16	.546
B5_Cons	*Conscientiousness.* See B5_Ag	5.14	1.09	5.18	1.09	.204	5.16	1.21	5.21	1.20	.151
B5_Extr	*Extraversion.* See B5_Ag.	4.33	1.32	4.35	1.31	.598	4.32	1.32	4.34	1.32	.600
B5_Neur	*Neuroticism.* See B5_Ag	3.61	1.39	3.61	1.42	1.0	3.52	1.38	3.48	1.38	.316
B5_Op	*Openness to Experience.* See B5_Ag	4.43	1.28	4.43	1.27	1.0	4.32	1.38	4.37	1.40	.212
Connect	*Perceived Connectedness.* Three items ranging from 1 (low) to 7 (high)	5.80	1.06	5.81	1.06	.744	5.83	1.07	5.84	1.07	.746
Decentering	*Decentering*. Five items capturing the extent to which people disengaged from thoughts and feelings from 1 (low) to 7 (high)	4.24	1.02	4.24	1.02	1.0	4.19	0.99	4.19	1.00	1.0

Name	Measurement	Participants (A)					Partners (B)				p-value
		pre		post			pre		post		
		mean	std	mean	std	p-value	mean	std	mean	std	
ECR	*Insecure Attachment.* Revised version of the *Experiences in Relationships Scale* (ECR-R) (Fraley et al., 2000)	2.24	0.99	2.18	0.96	.033	2.36	1.05	2.30	1.03	.046
ExCont	*Executive Control.* Four items from the *Deficits in Executive Functions Scale* (Barkley, 2011) from 1 (high) to 7 (low)	2.85	1.18	2.85	1.18	1.0	2.92	1.27	2.91	1.26	.784
FFMQ	*Mindfulness.* 24-item short version of the *Five Facet Mindfulness Questionnaire* (Bohlmeijer et al., 2011) from 1 (low) to 7 (high)	4.55	0.64	4.57	0.65	.282	4.50	0.66	4.54	0.67	.037
Stress	*Stress through Factors External to the Relationship.* Single item, ranged from 1 (no stress at all) to 7 (maximum stress).	3.38	1.63	3.36	1.63	.671	3.43	1.67	3.42	1.67	.836
RAS	*Relationship Satisfaction.* Seven item *Relationship Assessment Scale* (Hendrick, 1988) from 1 (low) to 7 (high)	5.85	1.00	5.86	1.00	.729	5.91	0.95	5.91	0.95	1.0
Rel_Years	*Relationship Duration in Years*. Rounded down to the next natural number (months and weeks are not considered)	22.02	14.59	22.76	14.77	.085	-	-	-	-	-
RDistress	*Relationship Distress.* Single item from 1 (none) to 7 (a lot)	3.38	1.63	2.51	1.33	.000	3.43	1.67	2.43	1.35	.000

Note. Complete list of the interval scaled data used pre and post deleting samples with missing values, including the standard deviation (std) and p-value from the t-test testing the pre and post distributions for statistical significance. Variables excluded for this analysis due to missing values: A_Selfesteem, B_Selfesteem, A_LifeSatisfaction, B_LifeSatisfaction*;* due to lack of comprehensibility: time_inc, time_post, int.fu, int.inc, int.post*;* due to doubling information: Rel_Months (relationship duration in months), group (intervention or control, ID_A, ID_B, ID_Member*;* due to too much information about A_RAS: GenRAS_A, GenRAS_B (general relationship satisfaction for A and B). Data kindly provided by Karremans et al. (2020).

Table 4

Descriptive Statistics of the Variables Cohabitation and Income

Name	Explanation	Yes		No		p-value
		pre	post	pre	post	
Cohabit	Yes = participant and partner share one household No = participant and partner live in separate households	92.92	93.41	7.08	6.59	.434
Intervention	Yes = participant did undertake a mindfulness intervention No = participant did not undertake a mindfulness intervention	51.47	53.45	48.53	46.55	.078

Note. Overview of the variables cohabit and intervention pre and post deleting samples with missing values, including the p-value from the binomial test testing the pre and post distributions for statistical significance. Data kindly provided by Karremans et al. (2020).

Table 3

Descriptive Statistics of the Variable Income

Name	Explanation	1			2			3			4			5			Missing Values		
		pre	post	p	pre	post	p	pre	post	p	pre	post	p	pre	post	p	pre	post	p
Income (euros per year)	1 = < 13.300 2 = 13.300 – 34.500 3 = 34.500 – 41.200 4 = 41.200 – 69.000 5 = > 69.000	1.82	2.09	.359	15.47	19.19	.000	21.23	25.97	.000	28.01	32.58	.000	18.00	20.17	.012	15.47	0	.000

Note. Overview of the income distribution in the sample, pre and post deleting samples and applying KNN-(k-Nearest-Neighbor)-imputation to the income categories, including the p-Value (p) from the binomial test testing the pre and post distributions for statistical significance. Data kindly provided by Karremans et al. (2020).

Table 4

Descriptive Statistics of the Variable Kids

Name	Explanation	Yes			No			NaN		
		pre	post	p-value	pre	post	p-value	pre	post	p-value
Kids	Yes = participant and partner have one or more children together No = participant and partner do not have children together NaN = prefer not to answer	72.20	72.58	.709	25.68	25.59	.939	2.12	1.83	.982

Note. Overview of the variable kids pre and post deleting samples with missing values, including the the p-value from the binomial test testing the pre and post distributions for statistical significance. Data kindly provided by Karremans et al. (2020).

Table 5

Descriptive Statistics of the Variable Sex

Name	Explanation	Participant Female		Participant Male		p-value	Partner Female		Partner Male		p-value
		pre	post	pre	post		pre	post	pre	post	
Sex	Sex separately for participants and partners, so out of 200%.	61.68	61.73	38.32	38.27	.982	37.71	37.73	62.29	62.27	.982

Note. Overview of the variable sex for participants and partners pre and post deleting samples with missing values, including the p-value from the binomial test testing the pre and post distributions for statistical significance. Only the categories “male” and “female” are listed, as in this study those were the only two sex/ gender categories used. Data kindly provided by Karremans et al. (2020)

8.3 Python Code

8.3.1 Script 1: Data Preparation

Script A

```
"""
Created on Wed Mar  8 10:39:50 2023

data prep and descriptives for thesis

@author: louleopold
"""

#%% importing packages
from pathlib import Path #because the pathnames give errors
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt # for feature importance
import seaborn as sns # for color palette in plots
from scipy.stats import ttest_ind_from_stats, binomtest
import os
#%% Data Download

#read file in and convert the pathname to spss file and make it a pandas dataframe
# uni pc : Path(r"C:\Users\leopoldm94\Desktop/msc\data.sav")
path_to_spss = Path("/Users/louleopold/Documents/Uni/Master
Psychologie/Masterarbeit/1-code/data.sav")
data = pd.read_spss(path_to_spss, usecols=None, convert_categoricals=True)

#set wd
os.chdir("/Users/louleopold/Documents/Uni/Master Psychologie/Masterarbeit/1-
code/descriptives")

#%% Descriptive Statistics before Data Prep

colors = sns.color_palette('colorblind')
# some single colors:
   # light blue '#a1c9f4'
   # light mint green '#8de5a1'
   # light salmon '#ff9f9b'
   # light purple "#d0bbff"
   # light yellow '#fffea3'
   # light turquoise '#b9f2f0'

#%%% Shape Pre Prep
print(f"the shape of the data before is {data.shape}")
```

```
#%%% Age

# make variables continous
data["A_Age"] = data["A_Age"].astype(float)
data["B_Age"] = data["B_Age"].astype(float)

# Get the age column from the DataFrame
A_Age = data['A_Age']
B_Age = data['B_Age']

#%%%% Age Distribution Table

# safe the basic distribution

A_Age_Before = data["A_Age"].describe()
B_Age_Before = data["B_Age"].describe()

Age_Before = pd.concat([A_Age_Before, B_Age_Before], axis=1)

#plt.figure(dpi = 2400)
# Create a figure and axis
fig, ax = plt.subplots()

# Create a table from the DataFrame and add it to the axis
table = ax.table(cellText=Age_Before.round(2).values, rowLabels = Age_Before.index,
colLabels=["Age Participants", "Age Partners"], loc='center')

table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1, 1.5)

# Hide the axis and grid
ax.axis('off')
ax.grid(False)

plt.title("Age Distribution in Data before dropping Samples",  size=12, y=0.9)

# Save the figure as a JPEG file
plt.savefig('Age_Before_Table.pdf', dpi=1200)

# Show the plot
plt.show()

#%%%% density plot age distribution


# Create a density plot of the age distribution
sns.kdeplot(A_Age, shade=True, label='Age Participants', color = colors[1] )
```

```
sns.kdeplot(B_Age, shade=True, label='Age Partners', color = colors[2])

# Add a title and labels to the plot
plt.title('Age Distribution Participants and their Partners')
plt.xlabel('Age')
plt.ylabel('Density')
plt.legend()

plt.savefig('Age_Before_DensityPlot.pdf', dpi=1200)

# Show the plot
plt.show()

#%%%% hist plot age

# Create a histogram of the age distribution
plt.hist([A_Age, B_Age], bins=20, color = [colors[1],colors[2]])

# Add a title and labels to the plot
plt.title('Age Distribution')
plt.xlabel('Age')
plt.ylabel('Frequency')
plt.legend(["Age Participants", "Age Partners"])

plt.savefig('Age_Before_Hist.pdf', dpi=1200)

# Show the plot
plt.show()

#%%% Sex

#%%%% Sex Distr Table

# safe the basic distribution

A_Sex_Female = data["A_Sex"].value_counts()["vrouw"]
A_Sex_Male = data["A_Sex"].value_counts()["man"]
B_Sex_Female = data["B_Sex"].value_counts()["vrouw"]
B_Sex_Male = data["B_Sex"].value_counts()["man"]

# Set the data for the table
Sex_Before = [[A_Sex_Female, A_Sex_Male],
      [B_Sex_Female, B_Sex_Male]]

# Set the row and column labels
row_labels = ["Participants", "Partners"]
column_labels = ["Female", "Male"]
```

```
# Create the table
fig, ax = plt.subplots()
ax.axis("off")

table = ax.table(cellText=Sex_Before, rowLabels=row_labels, colLabels=column_labels,
cellLoc="center", rowLoc="center", loc="center")
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1, 1.5)

# Hide the axis and grid
ax.axis('off')
ax.grid(False)

# Add a title for the table
plt.title("Sex Distribution Before Dropping Samples", size=12, y=0.7)

plt.savefig('Sex_Before_Table.pdf', dpi=1200)

plt.show()

#%%%% hist plot sex

# Create a histogram of the age distribution
plt.hist([data["A_Sex"], data["B_Sex"]], bins=2,color = [colors[1],colors[2]], rwidth=0.5)
plt.xticks([0.25, 0.75], labels=["Female", "Male"])

# Add a title and labels to the plot
plt.title('Sex Distribution')
plt.xlabel('Sex')
plt.ylabel('Frequency')
plt.legend(["Sex Participants", "Sex Partners"])

plt.savefig('Sex_Before_Hist.pdf', dpi=1200)

# Show the plot
plt.show()

#%%% hist plot income

data["A_Income"].replace("2 keer modaal of meer (€69.000 of meer)", 5, inplace = True)
data["A_Income"].replace("tussen 1 en 2 keer modaal (tussen €41.200 en €69.000)", 4,
inplace = True)
data["A_Income"].replace("modaal (tussen €34.500 en €41.200)", 3, inplace = True)
data["A_Income"].replace("beneden modaal (tussen €13.300 en €34.500)", 2, inplace =
True)
data["A_Income"].replace("minimum (minder dan €13.300)", 1, inplace = True)
data["A_Income"].replace("weet niet / wil ik niet zeggen", 0, inplace = True)
```

```
# Create a histogram of the age distribution
plt.hist(data["A_Income"], bins = 6, color = colors[1], rwidth = 0.75)
plt.xticks([0.4,1.25,2.1,2.95,3.8,4.6],["NaN",1,2,3,4,5])
# Add a title and labels to the plot
plt.title('Income Distribution Before dropping Samples and KNN Imputation')
plt.xlabel('Income Level')
plt.ylabel('Frequency')

plt.savefig('Income_Before_Hist.pdf', dpi=1200)

# Show the plot
plt.show()

#%%% B_Responsive

B_Responsive = data["B_Responsive"].describe()

print(B_Responsive)

# Create a histogram of the age distribution
plt.hist(data["B_Responsive"], bins=7,color = colors[1], rwidth=0.5)
#plt.xticks([0.25, 0.75], labels=["Female", "Male"])

# Add a title and labels to the plot
plt.title('Responsiveness of Partner')
plt.xlabel('B_Responsive')
plt.ylabel('Frequency')

plt.savefig('B_Responsive_Before_Hist.pdf', dpi=1200)

# Show the plot
plt.show()

# Create a density plot of the age distribution
sns.kdeplot(data["B_Responsive"], shade=True, color = colors[1])

# Add a title and labels to the plot
plt.title('Responsiveness of Partner')
plt.xlabel('B_Responsive')
plt.ylabel('Density')
plt.legend()

plt.savefig('B_Responsive_Before_DensityPlot.pdf', dpi=1200)

# Show the plot
plt.show()
```

```
#%%% Loop for Outcome and Features Table, no plots bc not numeric yet

features_to_plot = ["RAS", "Affect", "Rumi", "Stress", "RDistress", "Decentering", "ExCont",
"Connect", "Accept", "FFMQ", "ECR", "Commitment", "ECR", "Selfesteem", "B5_Ex",
"B5_Ag", "B5_Cons", "B5_Neur", "B5_Op"]

for i in features_to_plot:
   A_name = str("A_"+i)
   B_name = str("B_" +i)
   A = data[A_name]
   B = data[B_name]
   # Distribution Table
   # safe the basic distribution
   A_Describe = A.describe()
   B_Describe = B.describe()
   Before = pd.concat([A_Describe, B_Describe], axis=1)
   # Create a figure and axis
   fig, ax = plt.subplots()
   # Create a table from the DataFrame and add it to the axis
   table = ax.table(cellText=Before.round(2).values, rowLabels = Before.index,
colLabels=["Participants", "Partners"], loc='center')
   table.auto_set_font_size(False)
   table.set_fontsize(12)
   table.scale(1, 1.5)
      # Hide the axis and grid
   ax.axis('off')
   ax.grid(False)
   Title = str(i + " Distribution in Dataframe before dropping Samples")
   plt.title(Title,  size=12, y=0.9)
   # Save the figure as a JPEG file
   plt.savefig(str(i +'_Before_Table.pdf'), dpi = 1200)
   # Show the plot
   plt.show()

#%% Data Prep

#%%% replacing strings by numbers

# dummy coding sex
data["A_Sex"].replace("man", "A_male", inplace = True)
data["B_Sex"].replace("man", "B_male", inplace = True)
data["A_Sex"].replace("vrouw", "A_female", inplace = True)
data["B_Sex"].replace("vrouw", "B_female", inplace = True)
dummies_A_Sex = pd.get_dummies(data["A_Sex"])
dummies_B_Sex = pd.get_dummies(data["B_Sex"])

# dummy coding kids
```

```
data["A_Kids"].replace("nee", "have_no_kids", inplace = True)
data["A_Kids"].replace("ja, ik heb - kinderen (aantal)", "have_kids", inplace = True)
data["A_Kids"].replace("anders", "kids_NaN", inplace = True)
dummies_A_Kids = pd.get_dummies(data["A_Kids"])

# cohabit ja = 1 nee = 0
data["A_Cohabit"].replace("ja", "cohabit", inplace = True)
data["A_Cohabit"].replace("nee", "do_not_cohabit", inplace = True)
dummies_A_Cohabit = pd.get_dummies(data["A_Cohabit"])

# put all the dummies in the dataframe and remove originals later when other variables get
dropped
data = pd.concat([data, dummies_A_Sex, dummies_B_Sex, dummies_A_Kids,
dummies_A_Cohabit], axis = 1)
#data.drop("A_Sex", "B_Sex", "A_Kids", "A_Cohabit",inplace = True, axis=1)


# Affect: 1 for helemaal niet goed and 7 for heel er goed
data["A_Affect"].replace("helemaal niet goed", 1, inplace = True)
data["B_Affect"].replace("helemaal niet goed", 1, inplace = True)
data["A_Affect"].replace("heel erg goed", 7, inplace = True)
data["B_Affect"].replace("heel erg goed", 7, inplace = True)
# Rumi: 1 for helemaal niet van toepassing and 7 for heel erg van toepassing
data["A_Rumi"].replace("helemaal niet van toepassing", 1, inplace = True)
data["A_Rumi"].replace("heel erg van toepassing", 7, inplace = True)
data["B_Rumi"].replace("helemaal niet van toepassing", 1, inplace = True)
data["B_Rumi"].replace("heel erg van toepassing", 7, inplace = True)
# Stress: 1 for helemaal geen and 7 for heel veel
data["A_Stress"].replace("helemaal geen", 1, inplace = True)
data["A_Stress"].replace("heel veel", 7, inplace = True)
data["B_Stress"].replace("helemaal geen", 1, inplace = True)
data["B_Stress"].replace("heel veel", 7, inplace = True)
# Same for Distress
data["A_RDistress"].replace("helemaal geen", 1, inplace = True)
data["A_RDistress"].replace("heel veel", 7, inplace = True)
data["B_RDistress"].replace("helemaal geen", 1, inplace = True)
data["B_RDistress"].replace("heel veel", 7, inplace = True)
# Gen_RS: 1 for helemaal niet tevreden and 7 for heel erg tevreden (in case I want to use it)
data["GEN_RS_A"].replace("helemaal niet tevreden", 1, inplace = True)
data["GEN_RS_A"].replace("heel erg tevreden", 7, inplace = True)
data["GEN_RS_B"].replace("helemaal niet tevreden", 1, inplace = True)
data["GEN_RS_B"].replace("heel erg tevreden", 7, inplace = True)


#%%% dropping all variables that i do not need or do not understand
# MEMBER_ID bc thats already in id, time and int bc i do not understand these
# rel-months bc i do not think this is important, years are enough
# gen-rs-a bc i think it's almost same as target (can evaluate that tho)
```

```
# dropping groep, bc that's already in intervention
# dropping selfesteem bc don't know which is which

data = data.drop(["A_Sex", "B_Sex", "A_Kids", "A_Cohabit", 'groep', 'A_Selfesteem',
'MEMBER_ID', 'time_fu', 'time_inc', 'time_post', 'int.fu', 'int.inc', 'int.post', 'A_Rel_months',
'B_MEMBER_ID', 'A_MEMBER_ID', 'GEN_RS_A', "GEN_RS_B"], inplace = False, axis=1)

#%%% dropping NaNs

# Making a list of missing value types
missing_values = ["n/a", "na", "--", 999,9999, "weet niet / wil ik niet zeggen"]
data.replace(to_replace = missing_values, value = np.nan, inplace = True) # Replace values
data.isna().sum().sum() # 27037 with income part now 27496
pd.set_option('display.max_rows', None)
#pandas.set_option('display.max_rows', 10)
data.isna().sum()
# dropping the four variables with most missing values
data = data.drop(["B_Selfesteem", "A_Lifesat_pre", "B_Lifesat_pre"], axis = 1)
data.isna().sum().sum() # 18136 (still right after dropping Income aswell)
#data.shape # 2967, 62 (instead of 65 columns before), without Income now 61
data.dropna(inplace = True) # dropping NaNs
#data.shape # 2017, 61, so lost 950 samples
data.isna().sum().sum() # check categorical is complete, if any category is missing
#%% save data as csv
# save as csv for manipulation on other computers

data.to_csv('/Users/louleopold/Documents/Uni/Master Psychologie/Masterarbeit/1-
code/data-clean.csv', index=False)

#%% testing the data pre and post cleaning

# t test metrics participants

A_mean_1 =
[5.85,5.22,5.26,48.75,4.27,5.14,4.33,3.61,4.43,5.80,4.24,2.24,2.85,4.55,3.38,3.38]
A_sd_1 = [1.00,1.05,1.21,14.00,1.09,1.09,1.32,1.39,1.28,1.06,1.02,0.99,1.18,0.64,1.63,1.63]
A_mean_2 =
[5.86,5.23,5.27,49.40,4.29,5.18,4.35,3.61,4.43,5.81,4.24,2.18,2.85,4.57,3.36,2.51]
A_sd_2 = [1.00,1.05,1.21,14.10,1.11,1.09,1.31,1.42,1.27,1.06,1.02,0.96,1.18,0.65,1.63,1.33]
name = ["A_RAS/
B_RAS","Accept","Affect","Age","B5_Ag","B5_Cons","B5_Extr","B5_Neur","B5_Op","Connec
t","Decentering","ECR","ExCont","FFMQ","Stress","RDistress"]

p_A = []
for i in range(0, len(name)):
   test = ttest_ind_from_stats(A_mean_1[i], A_sd_1[i], 2967, A_mean_2[i], A_sd_2[i], 2017,
equal_var=True, alternative='two-sided')
   result = test.pvalue
```

```
    p_A.append(result)
    print(f"Participant {name[i]} p value {p_A[i]}")

# t test metrics partners

B_mean_1 =
[5.91,5.02,5.26,49.76,4.30,5.16,4.32,3.52,4.32,5.83,4.19,2.36,2.92,4.50,3.43,3.43]
B_sd_1 = [0.95,1.04,1.25,13.85,1.14,1.21,1.32,1.38,1.38,1.07,0.99,1.05,1.27,0.66,1.67,1.67]
B_mean_2 =
[5.91,5.03,5.27,50.50,4.32,5.21,4.34,3.48,4.37,5.84,4.19,2.30,2.91,4.54,3.42,2.43]
B_sd_2 = [0.95,1.04,1.25,13.90,1.16,1.20,1.32,1.38,1.40,1.07,1.00,1.03,1.26,0.67,1.67,1.35]

p_B = []
for i in range(0, len(name)):
    test = ttest_ind_from_stats(B_mean_1[i], B_sd_1[i], 2967, B_mean_2[i], B_sd_2[i], 2017,
equal_var=True, alternative='two-sided')
    result = test.pvalue
    p_B.append(result)
    print(f"Partner {name[i]} p value = {p_B[i]}")

# categorical variables

name = ["Cohabit", "Intervention", "Income1", "Income2", "Income3", "Income4",
"Income5", "IncomeNaN", "KidsYes", "KidsNo", "KidsNaN", "Gender Participant", "Gender
Partner"]
p_cat = [92.92,51.47,1.82,15.47,21.23,28.01,18,15.47,72.2,25.68,2.12,61.68,37.71]
p_cat_new = [93.41,53.45,2.09,19.19,25.97,32.58,20.17,0,72.58,25.59,1.83,61.73,37.73]
pvalue = []
for i in range(0, len(name)):
    test = binomtest(k= int(round((p_cat_new[i]*20.17),0)), n=2017, p=(p_cat[i]/100),
alternative='two-sided')
    result = test.pvalue
    pvalue.append(result)
    print(f"{name[i]} p value = {pvalue[i]}")
```

8.3.2 Script 2: Main Analysis

```
"""
Created on Fri Sep 30 09:21:18 2022

master thesis uni pc main version

@author: louleopold
"""

#%% importing packages
import numpy as np
import pandas as pd
```

```
from sklearn.preprocessing import StandardScaler
from matplotlib import pyplot as plt # for feature importance
import seaborn as sns # for color palette in plots
#from sklearn.inspection import permutation_importance
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
#from lineartree import LinearTreeRegressor
from lineartree import LinearForestRegressor
from sklearn.model_selection import GroupShuffleSplit, RandomizedSearchCV
#RepeatedStratifiedKFold ShuffleSplit
from sklearn.linear_model import Ridge # RidgeCV, LinearRegression
from time import time
from datetime import datetime
from sklearn.metrics import r2_score
from scipy.stats import uniform, loguniform
from scipy.stats import randint
from scipy.stats import t
import scipy.stats
import os
from sklearn.impute import KNNImputer
#%% Data Download and Prep
data = pd.read_csv(r'C:\Users\leopoldm94\Desktop/data-clean.csv')

# colors for graphs later later
colors = sns.color_palette('colorblind')

#%% CV

#%%% CV prep : defining variables and hyperparameters to tune

# define X, y and grouping
data["A_Income"].replace(0, np.nan, inplace = True)
X = data.drop(["A_RAS", "id"],axis=1).values
y = data['A_RAS'].values.reshape(-1,1)
g = data['id'].values.reshape(-1,1)

# parameters to tune in inner CV loop
param0 = {#'max_depth': np.arange(2,20, 1), # tiefe könnte man sparen, None als tiefe,
braucht man evtl aber mehr zeit
        'min_samples_leaf':randint(1,100),
         'min_samples_split': randint(2,200)}

param1 = {#'max_depth': randint(2,20), #linear trees akzeptiert kein None als tiefe
         'base_estimator__alpha': loguniform(1e-2, 1e1),
        'min_samples_leaf':randint(1,100),
         'min_samples_split': randint(2,200)} # wenn ich den auf 100 setze krieg ich ein LT r2
von -500, bei 200 von -30, ab 500 ist wieder alles normal.. obwohl min-sample-split wert
dann doch ur niedrig
```

```
param2 = {#'max_depth': np.arange(2,20, 1),
       'min_samples_leaf':randint(1,200), # wieder auf start 1 setzen
        'min_samples_split': randint(2,200),
        'max_features': uniform(0.01,0.99)}

param3 = {#'max_depth': randint(2,20),
        'base_estimator__alpha': loguniform(1e-2, 1e1),
       'min_samples_leaf':randint(1,100),
        'min_samples_split': randint(2,200),
        'max_features': uniform(0.01,0.99)}

# Declare the inner and outer cross-validation strategy
inner_cv = GroupShuffleSplit(n_splits=160, test_size=0.2) # 160 bc want 20000 samples and
have sample size of 2000*0.8*0.2 = 320, 2000%320=160
outer_cv = GroupShuffleSplit(n_splits=100, test_size=0.2, random_state=42) # 100


# define estimators
estimator0 = DecisionTreeRegressor()
estimator1 = LinearForestRegressor(base_estimator=Ridge(),  n_estimators=1,
bootstrap=False,
max_features=1.0)#LinearTreeRegressor(base_estimator=LinearRegression()),
base_estimator=RidgeCV(alphas = l2_alphas, cv = l2_cv)
estimator2 = RandomForestRegressor(n_estimators=300) #, max_features= 0.8 falls r2
wieder sehr schlecht
estimator3 = LinearForestRegressor(base_estimator=Ridge(),  n_estimators=200)
#base_estimator=RidgeCV(alphas= l2_alphas, cv = l2_cv)

# list of estimators and their parameters to loop thru
model = [estimator0, estimator1, estimator2, estimator3]
param = [param0, param1, param2, param3]
name = ["Decision Tree", "Linear Tree", "Random Forest", "Linear Forest"]

# Initialize empty list (to be filled with average results afterwards)
r2 = []

# Initialize nested list to save scores for plotting
# Initialize nested list to save scores for plotting
scores = [[]]*4
best_params = {0 : {}, 1 : {}, 2 : {}, 3 : {}}
mean_cv_score = [[]]*4
cv_output = {0 : {}, 1 : {}, 2 : {}, 3 : {}}
icv_scores = {0 : {}, 1 : {}, 2 : {}, 3 : {}}
run_times = []

#%%% CV Loop
```

```
# mit prep in  for schleife und liste der estimators, damits durchlaufen kann
# mehr tiefe als seperate param liste für nicht linear tree, bis max 50, evtl auch nur bis 30

for i in range(0, len(model)):
   t_start = time() # zwischenschritte ausgeben
   r2 = []
   j = 0
   for i_cv, (i_trn, i_tst) in enumerate(outer_cv.split(g, groups=g)):
      # split data02 into training and test set
      X_trn, y_trn, g_trn, X_tst, y_tst, g_tst = X[i_trn], y[i_trn], g[i_trn], X[i_tst], y[i_tst],
g[i_tst]
      #scaling with the test set as reference (bc linear models in lt and lf)
      scaler_features = StandardScaler(copy=True, with_mean=True,
with_std=True).fit(X_trn)
      scaler_targets = StandardScaler(copy=True, with_mean=True, with_std=True).fit(y_trn)
      scaler_groups = StandardScaler(copy=True, with_mean=True, with_std=True).fit(g_trn)
      # transform data to scaled version
      X_trn = scaler_features.transform(X_trn)
      y_trn = scaler_targets.transform(y_trn)
      g_trn = scaler_groups.transform(g_trn)
      X_tst = scaler_features.transform(X_tst)
      y_tst = scaler_targets.transform(y_tst)
      g_tst = scaler_groups.transform(g_tst)
      #knn
      imputer = KNNImputer()
      X_trn = imputer.fit_transform(X_trn)
      X_tst = imputer.fit_transform(X_tst)
      # inner cv
      search = RandomizedSearchCV(
         model[i],
         param[i],
         n_iter=100,
         scoring=None,
         n_jobs=-2,
         refit=True,
         cv=inner_cv,
         verbose=0,
         pre_dispatch='2*n_jobs',
         random_state=None,
         error_score=0,
         return_train_score=False) # jobs wie viele kerne werden belegt (max vier bei dem
laptop), plus anstecken
      result = search.fit(X_trn,y_trn.ravel(),groups = g_trn.ravel())
      # test regressor
      #best_params[i][j] = search.best_params_
      best_model = result.best_estimator_
      y_pred = best_model.predict(X_tst)
      # Calc metric (e.g. r2)
```

```
        current_r2 = r2_score(y_tst, y_pred)
        # Append to list
        r2.append(current_r2)
        icv_scores[i][j] = search.cv_results_
        best_params[i][j] = search.best_params_
        j += 1
    scores[i] = r2
    mean_cv_score[i] = np.mean(r2)
    cv_output[i]["mean_cv_score"] = np.mean(r2)
    cv_output[i]["std_cv_score"] = np.std(r2)
    cv_output[i]["median_cv_score"] = np.median(r2)
    cv_output[i]["params"] = search.best_params_
       #final: should be 100 values - 10 subsets *10 repeats
    print(f"The mean CV score of the best model is: {search.best_score_:.3f}")
    print(f"The mean score using nested cross-validation is: "
          f"{np.mean(r2):.3f} ± {np.std(r2):.3f}")
    print(f"The median score using nested cross-validation is: "
          f"{np.median(r2):.3f}")
    print('Elapsed time: '+str(np.round(
       time() - t_start, decimals=1)), end='\n\n')
    run_times.append(str(np.round(time() - t_start, decimals=1)))

#%%% save cv results

#%%%% prep

# save to specific folder C:\Users\leopoldm94\Desktop\output

os.chdir(r'C:\Users\leopoldm94\Desktop\output')

# create individual name

timestr = datetime.now().strftime("%Y-%m-%d_%H-%M")

name = timestr + "__ocv_" + str(outer_cv.n_splits) + "__icv_" + str(inner_cv.n_splits)

DT_scores = np.array(scores[0])
LT_scores = np.array(scores[1])
RF_scores = np.array(scores[2])
LF_scores = np.array(scores[3])

#%%%% save r2 scores and icv as excel

excel_cv_statistics = pd.DataFrame({
   "DT_mean" : cv_output[0]["mean_cv_score"], "DT_std" : cv_output[0]["std_cv_score"],
"DT_median": cv_output[0]["median_cv_score"] ,
   "LT_mean" : cv_output[1]["mean_cv_score"], "LT_std" : cv_output[1]["std_cv_score"],
"LT_median": cv_output[1]["median_cv_score"] ,
```

```
   "RF_mean" : cv_output[2]["mean_cv_score"], "RF_std" : cv_output[2]["std_cv_score"],
"RF_median": cv_output[2]["median_cv_score"] ,
   "LF_mean" : cv_output[3]["mean_cv_score"], "LF_std" : cv_output[3]["std_cv_score"],
"LF_median": cv_output[3]["median_cv_score"]},
   columns = ["DT_mean", "DT_std", "DT_median",
           "LT_mean", "LT_std", "LT_median",
           "RF_mean", "RF_std", "RF_median",
           "LF_mean", "LF_std", "LF_median"],
   index = np.arange(0, 1))

cv_statistics_name = "cv-statistics"
excel_cv_statistics.to_excel(name + "__" + cv_statistics_name + ".xlsx")

# single estimator r2 cv scores

excel_cv_scores = pd.DataFrame({"DT_scores" : DT_scores, "LT_scores" : LT_scores,
                  "RF_scores" : RF_scores, "LF_scores" : LF_scores},
                 columns = ["DT_scores", "LT_scores", "RF_scores", "LF_scores"],
                 index = np.arange(0, len(DT_scores)))

cv_scores_name = "cv-scores"
excel_cv_scores.to_excel(name + "__" + cv_scores_name + ".xlsx")

# params

full_cv_output = pd.DataFrame(cv_output)
full_cv_output.to_excel( name + "__full-cv-output" +".xlsx")

full_icv_output = pd.DataFrame(icv_scores)
full_icv_output.to_excel( name + "__full-icv-output" +".xlsx")


# run times

run_times = pd.DataFrame(run_times)
run_times.to_excel( name + "__run_times" +".xlsx")

# best params

best_params = pd.DataFrame(best_params)
best_params.to_excel( name + "__best_params" +".xlsx")
#%%%  boxplot

# boxplots
for i in range(0, len(model)):
   colors = sns.color_palette('colorblind')
   #plt.subplot(1, 4, i+1)
   bp = plt.boxplot(scores, showmeans=(True), meanline=(True), patch_artist=colors[i])
```

```
    plt.title("Comparision of the R² Scores of the Different Estimators", fontsize = 12)
    plt.ylabel('Estimators')
    plt.ylabel('R² Scores')
    plt.xticks([1, 2, 3, 4], ["DT", "LT", "RF", "LF"])
    for patch, color in zip(bp['boxes'], colors):
        patch.set_facecolor(color)
plt.savefig(name + "__" + "boxplot" + ".jpg", dpi = 1200)
plt.show()
#%% f tests

def f_test(group1, group2):
    f = np.var(group1, ddof=1)/np.var(group2, ddof=1)
    nun = group1.size - 1
    dun = group2.size - 1
    p_value = 1 - scipy.stats.f.cdf(f, nun, dun)
    return f, p_value

f_test(DT_scores, LT_scores)
f_test(LT_scores, RF_scores)
f_test(RF_scores, LF_scores)

#%% t test

#%%% defining t-test functions
# imported t from scipy stats
# code from https://scikit-
learn.org/stable/auto_examples/model_selection/plot_grid_search_stats.html

def corrected_std(differences, n_train, n_test):
    """Corrects standard deviation using Nadeau and Bengio's approach.

    Parameters
    ----------
    differences : ndarray of shape (n_samples,)
        Vector containing the d ifferences in the score metrics of two models.
    n_train : int
        Number of samples in the training set.
    n_test : int
        Number of samples in the testing set.

    Returns
    -------
    corrected_std : float
        Variance-corrected standard deviation of the set of differences.
    """
    # kr = k times r, r times repeated k-fold crossvalidation,
    # kr equals the number of times the model was evaluated
    kr = len(differences)
```

```
    corrected_var = np.var(differences, ddof=1) * (1 / kr + n_test / n_train)
    corrected_std = np.sqrt(corrected_var)
    return corrected_std


def compute_corrected_ttest(differences, df, n_train, n_test):
    """Computes right-tailed paired t-test with corrected variance.

    Parameters
    ----------
    differences : array-like of shape (n_samples,)
        Vector containing the differences in the score metrics of two models.
    df : int
        Degrees of freedom.
    n_train : int
        Number of samples in the training set.
    n_test : int
        Number of samples in the testing set.

    Returns
    -------
    t_stat : float
        Variance-corrected t-statistic.
    p_val : float
        Variance-corrected p-value.
    """
    mean = np.mean(differences)
    std = corrected_std(differences, n_train, n_test)
    t_stat = mean / std
    p_val = t.sf(np.abs(t_stat), df)  # right-tailed t-test
    return t_stat, p_val

#%%% t-test prep

n_train = X_trn.shape[0]
n_test = X_tst.shape[0]

#%%% LT - DT

differences_trees = LT_scores - DT_scores
n_trees = differences_trees.shape[0]  # number of test sets
df_trees = n_trees - 1

t_stat_trees, p_val_trees = compute_corrected_ttest(differences_trees, df_trees, n_train,
n_test)
print(f"Corrected t-value trees: {t_stat_trees:.3f}\nCorrected p-value trees:
{p_val_trees:.3f}")
```

```
#%%% LF - RF

differences_forests = LF_scores - RF_scores
n_forests = differences_forests.shape[0]
df_forests = n_forests - 1

t_stat_forests, p_val_forests = compute_corrected_ttest(differences_forests, df_forests,
n_train, n_test)
print(f"Corrected t-value forests: {t_stat_forests:.3f}\nCorrected p-value forests:
{p_val_forests:.3f}")

#%%% LT - RF

differences_other = LT_scores - RF_scores
n_other = differences_other.shape[0]
df_other = n_other - 1

t_stat_other, p_val_other = compute_corrected_ttest(differences_other, df_other, n_train,
n_test)
print(f"Corrected t-value lt-rf: {t_stat_other:.3f}\nCorrected p-value lt-rf: {p_val_other:.3f}")

#%%% save ttest Outputs

# save t-test results in excel

excel_ttest_results = pd.DataFrame({"t_stat_trees" : t_stat_trees, "p_val_trees" :
p_val_trees,
                    "t_stat_forests" : t_stat_forests, "p_val_forests" : p_val_forests,
                    "t_stat_other" : t_stat_other, "p_val_other" : p_val_other},
                   columns = ["t_stat_trees", "p_val_trees",
                           "t_stat_forests", "p_val_forests",
                           "t_stat_other", "p_val_other"],
                   index = np.arange(0, 1))

ttest_results_name = "t-test_results"
excel_ttest_results.to_excel(name + "__" + ttest_results_name + ".xlsx")


#%%% ttest boxplot

# boxplots
for i in range(0, len(model)):
   colors = sns.color_palette('colorblind')
   #plt.subplot(1, 4, i+1)
   bp = plt.boxplot(scores, showmeans=(True), meanline=(True), patch_artist=colors[i])
   plt.title("Comparision of the R² Scores of the Different Estimators", fontsize = 12)
   plt.ylabel('Estimators')
   plt.ylabel('R² Scores')
```

```
    plt.xticks([1, 2, 3, 4], ["DT", "LT", "RF", "LF"])
    for patch, color in zip(bp['boxes'], colors):
        patch.set_facecolor(color)
# significance bars
p_values = (p_val_trees, p_val_other, p_val_forests)
for j in range(3):
    A = j+1
    B = j+2
    y_pos_A = max(scores[j]) * 1.005
    y_pos_B = max(scores[j+1]) * 1.005
    x_pos_A = j+ 1.05
    x_pos_B = j+ 1.95
    bar_height = max(y_pos_A, y_pos_B) * 1.01
    text_height = bar_height - 0.02
    text = str("p = "+ str(p_values[j].round(3)))
    plt.plot([x_pos_A, x_pos_A, x_pos_B ,x_pos_B], [y_pos_A , bar_height, bar_height,
y_pos_B], linewidth=1, color="black")
    plt.text((j +1 + j +2) * 0.5, text_height, text, ha='center', va='bottom', c="black")
plt.savefig(name + "__" + "boxplot_T-Test_" + ".jpg", dpi = 1200)
plt.show()

# results table

# create a list of the estimator names
estimator_names = ['DT vs. LT', 'LT vs. RF', 'RF vs. LF']

# create a list of the t test scores for each estimator
t_scores = [t_stat_trees, t_stat_other, t_stat_forests]

# create a list of the p values for each estimator
p_values = [p_val_trees, p_val_other, p_val_forests]

# create an array of the data
t_test_results_table_data = np.array([t_scores, p_values])

# create a figure and axis
fig, ax = plt.subplots()

# create the table with the data and estimator names as row labels
table = ax.table(cellText=t_test_results_table_data.round(3), rowLabels=['T-Score', 'P-
Value'], colLabels=estimator_names, loc='center')

# adjust the appearance of the table
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1, 1.5)

# Hide the axis and grid
```

```
ax.axis('off')
ax.grid(False)

plt.title("T-Test Results (with Nadeau Bengio Correction",  size=12, y=0.7)

# Save the figure as a JPEG file
plt.savefig('T-Test_Results_Table.pdf', dpi=1200)

# show the plot
plt.show()
#%% Welch

#%%% Defs

# from Davids plots script

def corr_std_error(x, tst_size_frac):
    """Corrects standard error using Nadeau and Bengio's approach.
    Ref: Nadeau, C., Bengio, Y. Inference for the Generalization Error.
    Machine Learning 52, 239–281 (2003).
    https://doi.org/10.1023/A:1024068626366

    Parameters
    ----------
    x : array of float
        Cross validation result data.
    tst_size_frac : float
        Fraction of samples in the testing set.

    Returns
    -------
    corr_std_error : float
        Variance-corrected standard error of the data x.
    """
    # Compute variance of x
    var_x = np.var(x)
    # Compute corrected standard error
    corr_std_error = np.sqrt(var_x/len(x) +
                             var_x*tst_size_frac/(1-tst_size_frac))
    # Return corrected std
    return corr_std_error


def dep_two_sample_ttest(x1, x2, tst_size_frac, side='two'):
    """
    Implementation of the Nadeau and Bengio correction of dependent sample
    (due to cross-validation's resampling) two sample Student's t-test for
    unequal sample sizes and unequal variances aka Welch's test
```

```
t_stat = mean(x1)-mean(x2)/sqrt(std_error(x1)²+std_error(x2)²)
whereas std_error(x) (= std(x)/sqrt(n_samples(x))) is replaced by
corrected_std_error (= corrected_std(x)/sqrt(n_samples(x)))
Ref: Nadeau, C., Bengio, Y. Inference for the Generalization Error.
Machine Learning 52, 239–281 (2003).
https://doi.org/10.1023/A:1024068626366
Ref: https://scikit-learn.org/stable/auto_examples/model_selection/
plot_grid_search_stats.html

Parameters
----------
x1 : ndarray
   Data of condition 1.
x2 : ndarray
   Data of condition 2.
tst_size_frac : float
   Test size fraction.
side : sring ('one', 'two')
   If test is one or two sided. Default is two.

Returns
-------
t_stat : float
   Corrected t statistic.
df : float
   Degrees of freedom.
p : float
   Corrected p value.

"""
# Get n samples data 1
n1 = len(x1)
# Get n samples data 2
n2 = len(x2)
# Get corrected standard error of data 1
std_error_x1 = corr_std_error(x1, tst_size_frac)
# Get corrected standard error of data 2
std_error_x2 = corr_std_error(x2, tst_size_frac)
# Compute corrected t statistics
t_stat = (np.mean(x1)-np.mean(x2))/np.sqrt(std_error_x1**2+std_error_x2**2)
# Compute standard deviation of data 1
std_dev_x1 = std_error_x1*np.sqrt(n1)
# Compute standard deviation of data 2
std_dev_x2 = std_error_x2*np.sqrt(n2)
# degrees of freedom
df = (std_dev_x1**2/n1+std_dev_x2**2/n2)**2/(
   std_dev_x1**4/(n1**2 * (n1-1)) + std_dev_x2**4/(n2**2 * (n2-1)))
# calculate the p-value
```

```
    if 'one' in side:
        p_val = t.sf(abs(t_stat), df)
    elif 'two' in side:
        p_val = t.sf(abs(t_stat), df) * 2.0
    else:
        print('Error: unknown side')
    # return everything
    return t_stat, df, p_val

#%%% Prep

DT_scores = np.array(scores[0])
LT_scores = np.array(scores[1])
RF_scores = np.array(scores[2])
LF_scores = np.array(scores[3])

n_train = X_trn.shape[0]
n_test = X_tst.shape[0]

#%%% Trees

trees_t_stat, trees_df, trees_p_val = dep_two_sample_ttest(DT_scores, LT_scores, 0.2)
print(f"Corrected Welch's t-value dt-lt: {trees_t_stat:.3f}\nCorrected Welch's p-value dt-lt:
{trees_p_val:.3f}")

#%%% Forests

forests_t_stat, forests_df, forests_p_val = dep_two_sample_ttest(RF_scores, LF_scores, 0.2)
print(f"Corrected Welch's t-value rf-lf: {forests_t_stat:.3f}\nCorrected Welch's p-value rf-lf:
{forests_p_val:.3f}")

#%%% Other

other_t_stat, other_df, other_p_val = dep_two_sample_ttest(LT_scores, RF_scores, 0.2)
print(f"Corrected Welch's t-value lt-rf: {other_t_stat:.3f}\nCorrected Welch's p-value lt-rf:
{other_p_val:.3f}")

#%%% save welchs ttest to excel
# t-test results

excel_welchs_ttest_results = pd.DataFrame({"t_stat_trees" : trees_t_stat, "p_val_trees" :
trees_p_val,
                    "t_stat_forests" : forests_t_stat, "p_val_forests" : forests_p_val,
                    "t_stat_other" : other_t_stat, "p_val_other" : other_p_val},
                   columns = ["t_stat_trees", "p_val_trees",
                              "t_stat_forests", "p_val_forests",
                              "t_stat_other", "p_val_other"],
                   index = np.arange(0, 1))
```

```
welchs_ttest_results_name = "welchs_t-test_results"
excel_welchs_ttest_results.to_excel(name + "__" + welchs_ttest_results_name + ".xlsx")

#%%% welchs ttest boxplot
# boxplot with results

# boxplots
for i in range(0, len(model)):
   colors = sns.color_palette('colorblind')
   #plt.subplot(1, 4, i+1)
   bp = plt.boxplot(scores, showmeans=(True), meanline=(True), patch_artist=colors[i])
   plt.title("Comparision of the R² Scores of the Different Estimators", fontsize = 12)
   plt.ylabel('Estimators')
   plt.ylabel('R² Scores')
   plt.xticks([1, 2, 3, 4], ["DT", "LT", "RF", "LF"])
   for patch, color in zip(bp['boxes'], colors):
      patch.set_facecolor(color)
# significance bars
p_values = (trees_p_val, other_p_val, forests_p_val)
for j in range(3):
   A = j+1
   B = j+2
   y_pos_A = max(scores[j]) * 1.005
   y_pos_B = max(scores[j+1]) * 1.005
   x_pos_A = j+ 1.05
   x_pos_B = j+ 1.95
   bar_height = max(y_pos_A, y_pos_B) * 1.01
   text_height = bar_height - 0.02
   text = str("p = "+ str(p_values[j].round(3)))
   plt.plot([x_pos_A, x_pos_A, x_pos_B ,x_pos_B], [y_pos_A , bar_height, bar_height,
y_pos_B], linewidth=1, color="black")
   plt.text((j +1 + j +2) * 0.5, text_height, text, ha='center', va='bottom', c="black")
plt.savefig(name + "__" + "boxplot_Welchs_T-Test_" + ".jpg", dpi = 1200)
plt.show()

# results table

# create a list of the estimator names
estimator_names = ['DT vs. LT', 'LT vs. RF', 'RF vs. LF']

# create a list of the t test scores for each estimator
t_scores = [trees_t_stat, other_t_stat, forests_t_stat]

# create a list of the p values for each estimator
p_values = [trees_p_val, other_p_val, forests_p_val]

# create an array of the data
```

```
t_test_results_table_data = np.array([t_scores, p_values])

# create a figure and axis
fig, ax = plt.subplots()

# create the table with the data and estimator names as row labels
table = ax.table(cellText=t_test_results_table_data.round(3), rowLabels=['T-Score', 'P-
Value'], colLabels=estimator_names, loc='center')

# adjust the appearance of the table
table.auto_set_font_size(False)
table.set_fontsize(12)
table.scale(1, 1.5)

# Hide the axis and grid
ax.axis('off')
ax.grid(False)

plt.title("T-Test Results (Welch's T-Test with Nadeau Bengio Correction",  size=12, y=0.7)

# Save the figure as a JPEG file
plt.savefig('Welchs_T-Test_Results_Table.pdf', dpi=1200)

# show the plot
plt.show()

#%% Outlook

#%%% Hyperparameter Tuning Graphs

#%%%% Prep

# list of r2 scores for each estimator in the inner cv, length of n_iter

icv_mean_test_scores = [[]]*4

for i in range(0, len(model)):
   icv_mean_test_scores[i] = []
   for j in range(0, outer_cv.n_splits):
      for element in icv_scores[i][j]["mean_test_score"]:
         icv_mean_test_scores[i].append(element)

# list of tried min_samples_leaf scores for each estimator in the inner cv, length of n_iter

icv_min_samples_leaf = [[]]*4

for i in range(0, len(model)):
   icv_min_samples_leaf[i] = []
```

```
    for j in range(0, outer_cv.n_splits):
        for h in range(0, search.n_iter):
            icv_min_samples_leaf[i].append(icv_scores[i][j]["params"][h]["min_samples_leaf"])

# list of tried min_samples_split scores for each estimator in the inner cv, length of n_iter

icv_min_samples_split = [[]]*4

for i in range(0, len(model)):
    icv_min_samples_split[i] = []
    for j in range(0, outer_cv.n_splits):
        for h in range(0, search.n_iter):
            icv_min_samples_split[i].append(icv_scores[i][j]["params"][h]["min_samples_split"])

# list of tried base_estimator__alpha scores for each estimator in the inner cv, length of
n_iter

icv_base_estimator__alpha = [[]]*4

for i in [1,3]:
    icv_base_estimator__alpha[i]=[]
    for j in range(0, outer_cv.n_splits):
        for h in range(0, search.n_iter):

icv_base_estimator__alpha[i].append(icv_scores[i][j]["params"][h]["base_estimator__alpha
"])

# list of tried max_features scores for each estimator in the inner cv, length of n_iter

icv_max_features = [[]]*4

for i in [2,3]:
    icv_max_features[i]=[]
    for j in range(0, outer_cv.n_splits):
        for h in range(0, search.n_iter):
            icv_max_features[i].append(icv_scores[i][j]["params"][h]["max_features"])




#%%%% Scatter Plots

# Min Samples Leaf

for i in range(0, len(model)):
    plt.scatter(icv_min_samples_leaf[i],icv_mean_test_scores[i], color=colors[i], alpha = 0.01)
    plt.title("Tuning of the Hyperparemeter Min_Samples_Leaf", fontsize = 12)
    plt.xlabel('Min Samples Leaf')
```

```
    plt.ylabel('R² Scores')
    plt.legend(["DT", "LT","RF","LF"])
plt.savefig(name + "__min-samples-leaf_Scatter" + ".jpg", dpi = 1200)
plt.show()

# Min Samples Split

for i in range(0, len(model)):
    plt.scatter(icv_min_samples_split[i], icv_mean_test_scores[i], color=colors[i], alpha =
0.01)
    plt.title("Tuning of the Hyperparemeter Min_Samples_Split", fontsize = 12)
    plt.xlabel('Min Samples Split')
    plt.ylabel('R² Scores')
    plt.legend(["DT", "LT","RF","LF"])
plt.savefig(name + "__min-samples-split_Scatter" + ".jpg", dpi = 1200)
plt.show()

# base_estimator__alpha

for i in [1,3]:
    plt.scatter(icv_base_estimator__alpha[i], icv_mean_test_scores[i], color=colors[i], alpha =
0.01)
    plt.title("Tuning of the Hyperparemeter Lambda", fontsize = 12)
    plt.xlabel('Lambda')
    plt.ylabel('R² Scores')
    plt.legend(["LT","LF"])
plt.savefig(name + "__Lambda_Scatter" + ".jpg", dpi = 1200)
plt.show()

# max_features

for i in [2,3]:
    plt.scatter(icv_max_features[i], icv_mean_test_scores[i], color=colors[i]
            , alpha = 0.01)
    plt.title("Tuning of the Hyperparemeter Max Features", fontsize = 12)
    plt.xlabel('Max Features')
    plt.ylabel('R² Scores')
    plt.legend(["RF","LF"])
plt.savefig(name + "__max-features_Scatter" + ".jpg", dpi = 1200)
plt.show()

# list of tried min_samples_leaf scores for each estimator in the inner cv, length of n_iter
#%%%% Prep historgram

msl = [[]]*4

for i in range(0, len(model)):
    msl[i] = []
```

```
    for j in range(0, outer_cv.n_splits):
        msl[i].append(best_params[i][j]["min_samples_leaf"])

mss = [[]]*4

for i in range(0, len(model)):
    mss[i] = []
    for j in range(0, outer_cv.n_splits):
        mss[i].append(best_params[i][j]["min_samples_split"])

lambdaa = [[]]*4

for i in [1,3]:
    lambdaa[i] = []
    for j in range(0, outer_cv.n_splits):
        lambdaa[i].append(best_params[i][j]["base_estimator__alpha"])

maxfeat = [[]]*4

for i in [2,3]:
    maxfeat[i] = []
    for j in range(0, outer_cv.n_splits):
        maxfeat[i].append(best_params[i][j]["max_features"])

#%%%% Histograms

# Min Samples Leaf

plt.hist(msl)
plt.title("Tuning of the Hyperparemeter Min_Samples_Leaf", fontsize = 12)
plt.xlabel('Min Samples Leaf')
plt.legend(["DT", "LT","RF","LF"])
plt.savefig(name + "__min-samples-leaf_Hist" + ".jpg", dpi = 1200)
plt.show()

# Min Samples Split
plt.hist(mss)
plt.title("Tuning of the Hyperparemeter Min_Samples_Split", fontsize = 12)
plt.xlabel('Min Samples Split')
plt.legend(["DT", "LT","RF","LF"])
plt.savefig(name + "__min-samples-split_Hist" + ".jpg", dpi = 1200)
plt.show()

# base_estimator__alpha
plt.hist(lambdaa)
plt.title("Tuning of the Hyperparemeter Lambda", fontsize = 12)
plt.xlabel('Lambda')
plt.legend(["DT", "LT","RF","LF"])
```

```
plt.savefig(name + "__lambda_Hist" + ".jpg", dpi = 1200)
plt.show()

# max_features
plt.hist(maxfeat)
plt.title("Tuning of the Hyperparemeter Max_features", fontsize = 12)
plt.xlabel('Max_Features')
plt.legend(["DT", "LT","RF","LF"])
plt.savefig(name + "__maxfeat_Hist" + ".jpg", dpi = 1200)
plt.show()


#%%% Compare N-Estimators

#%%%% prep
n_est_mean_r2 = [[]]*2


parameters1 = {#'max_depth': np.arange(2,20, 1),
       'min_samples_leaf':randint(1,200), # wieder auf start 1 setzen
        'min_samples_split': randint(2,200),
        'max_features': uniform(0.01,0.99)}

parameters2 = {#'max_depth': randint(2,20),
        'base_estimator__alpha': loguniform(1e-2, 1e1),
       'min_samples_leaf':randint(1,100),
        'min_samples_split': randint(2,200),
        'max_features': uniform(0.01,0.99)}

parameters = (parameters1, parameters2)

# Declare the inner and outer cross-validation strategy
inner_cv = GroupShuffleSplit(n_splits=160, test_size=0.2) # 160 bc want 20000 samples and
have sample size of 2000*0.8*0.2 = 320, 2000%320=160
outer_cv = GroupShuffleSplit(n_splits=100, test_size=0.2, random_state=42) # 100

#%%%% cv
for i in [0,1]:
   t_start = time() # zeit für zwischenschritte ausgeben
   n_est_mean_r2[i] = []
   counter = 0
   for j in [10,20,50,100,200,300]:
      r2 = []
      score = []
      forests =
(RandomForestRegressor(n_estimators=j),LinearForestRegressor(base_estimator=Ridge(),
n_estimators=j) )
      name = ("random forest", "linear forest")
```

```
    for i_cv, (i_trn, i_tst) in enumerate(outer_cv.split(g, groups=g)):
        X_trn, y_trn, g_trn, X_tst, y_tst, g_tst = X[i_trn], y[i_trn], g[i_trn], X[i_tst], y[i_tst],
g[i_tst]
        scaler_features = StandardScaler(copy=True, with_mean=True,
with_std=True).fit(X_trn)
        scaler_targets = StandardScaler(copy=True, with_mean=True,
with_std=True).fit(y_trn)
        scaler_groups = StandardScaler(copy=True, with_mean=True,
with_std=True).fit(g_trn)
        X_trn = scaler_features.transform(X_trn)
        y_trn = scaler_targets.transform(y_trn)
        g_trn = scaler_groups.transform(g_trn)
        X_tst = scaler_features.transform(X_tst)
        y_tst = scaler_targets.transform(y_tst)
        g_tst = scaler_groups.transform(g_tst)
        imputer = KNNImputer()
        X_trn = imputer.fit_transform(X_trn)
        X_tst = imputer.fit_transform(X_tst)
        search = RandomizedSearchCV(
          forests[i],
          parameters[i],
          n_iter=100,
          scoring=None,
          n_jobs=-2,
          refit=True,
          cv=inner_cv,
          verbose=0,
          pre_dispatch='2*n_jobs',
          random_state=None,
          error_score=0,
          return_train_score=False)
        result = search.fit(X_trn,y_trn.ravel(),groups = g_trn.ravel())
        best_model = result.best_estimator_
        y_pred = best_model.predict(X_tst)
        current_r2 = r2_score(y_tst, y_pred)
        r2.append(current_r2)
        score.append(np.mean(r2))
    n_est_mean_r2[i].append(np.mean(score))
    counter += 1
    print(f"The {counter}th mean r2 for {name[i]} found is: "
          f"{np.mean(r2):.3f} ± {np.std(r2):.3f}")
    print('Elapsed time: '+str(np.round(
        time() - t_start, decimals=1)), end='\n\n')

for i in range(0,2):
  plt.plot([10,20,50,100,200,300], n_est_mean_r2[i], color= colors[i+2])
  plt.title("Impact of N Estimators", fontsize = 12)
  plt.xlabel('N Estimators')
```

```
    plt.xscale("log")
    plt.xticks([10,20,50,100,200,300],[10,20,50,100,200,300])
    plt.ylabel('R² Scores')
    plt.legend(["RF","LF"])
plt.savefig(name+"__N-Estimators_Line" + ".jpg", dpi = 1200)
```

9 Acknowledgements

I would like to express my sincere gratitude to my supervisor Prof. Frank Scharnowski, for his guidance and invaluable insights throughout the course of my master's thesis. His expertise and continuous support empowered me to pursue my studies with determination.

I am also deeply indebted to my co-supervisor, Dr. David Steyrl, for all of his invaluable contributions, countless hours of insightful discussion and advice, almost endless patience, and tireless dedication to improving the outcome of this thesis. His expertise and exceptional mentorship were essential in overcoming challenges, reaching meaningful conclusions, and shaping the direction and quality of this thesis and my mind.

I extend my sincere appreciation to Dr. David Steyrl for generously sharing his code, which contributed significantly to the implementation and success of the comparative analysis.

I would like to express my gratitude to Marco Cerliani for sharing his code on linear trees on github, which served as an essential foundation for my analysis and saved me from many hours of tears, sweat, and anger.

I would like to thank my sister Ann-Katrin Leopold and my friends Jessica Angioni, Carmen Baskauf, Marie Bechter, Laura Coral, Paul Eberle, Simon Faßnacht, Robert Geppert and Roxana Pohlmann for their invaluable help in proofreading and providing feedback on my thesis. Their critical evaluation and constructive suggestions have greatly improved the clarity and coherence of this document.

I owe gratitude to my parents for their support and encouragement. Their sacrifices throughout my academic journey have enabled me to pursue my studies with determination.

I would like to express my heartfelt appreciation to my roommates, whose confidence in my abilities and support were beyond words. Their understanding, encouragement, and delicious meals provided much needed comfort and nourishment, allowing me to focus on my studies.

Finally, I am deeply grateful to all of my friends for their constant encouragement, love, and unwavering support. Their presence and unwavering faith in my potential has been a constant source of inspiration and motivation.

I am truly fortunate to have had the privilege of working with and receiving support from such extraordinary individuals. Their contributions and encouragement have played a significant role in the successful completion of this Master's thesis.