OULUN AMMATTIKORKEAKOULU

THE IMPACT OF ACCESSIBILITY OF EDUCATION ON YOUTH UNEMPLOYMENT

A prediction model using machine learning

Peter Forsell
Master's Thesis
Autumn 2025
Master of Engineering in Data Analytics
Oulu University of Applied Sciences

Man muß noch Chaos in sich haben, um einen tanzenden Stern gebären zu können.

(Nietzsche, F. 1883, 15.)

To my daughter Nea

with love,

Dad.

Ever tried. Ever failed. No matter. Try again. Fail again. Fail better.

(Beckett, S. 1983, 7.)

ABSTRACT

Oulu University of Applied Sciences
Master of Engineering in Data Analytics

Author(s): Peter Forsell
Title of thesis: The Impact of Accessibility of Education on Youth Unemployment
Thesis supervisor(s): Anu Niva
Term and year of completion: Autumn 2025
Pages: e.g. 64 + 8 appendices

This thesis examined whether municipal access to upper-secondary education is associated with youth unemployment in Finland. The work was motivated by regional disparities in study opportunities and labour-market outcomes, and by current policy debates on vocational education. The aim was to estimate the strength and direction of this association and to test the suitability of common machine learning methods for municipal-level prediction.

A combined dataset was built from official Finnish sources: education institution counts by type, municipal population totals, land-area data, and youth-unemployment figures. After harmonising municipality names, population density and institutions per 10,000 residents were derived, and youth unemployed per 10,000 residents was defined as the target. Basic cleaning and outlier screening were carried out.

Two regression models were trained on the same features. A k-nearest neighbours model served as a simple, assumption-light baseline; a random forest was used to model possible non-linear and interaction effects. Hyperparameters were tuned by cross-validation, and performance was evaluated on a held-out test set.

Both models found only a weak negative association between education-network density and youth unemployment once population size and density were controlled for. Predictive accuracy stayed modest, although the random forest improved on k-NN. It is concluded that educational accessibility contains signal but does not, on its own, explain municipal differences; further gains will require adding socio-economic, spatial, and labour-demand variables rather than increasing model complexity.

CONTENTS

GLOSSARY

BAGGING	Bootstrap Aggregating is a machine learning technique used in random forest, where multiple models are trained on random subsets of data to improve accuracy.
BLACK BOX	A "black box" in machine learning refers to a model that makes predictions without providing a clear or easily understandable explanation of how it arrives at those predictions.
BS	Bootstrap Sampling is a method of generating multiple samples from a dataset by random selection with replacement, used in random forest.
BVT	Bias-Variance Tradeoff is a fundamental problem in machine learning that involves balancing underfitting (high bias) and overfitting (high variance).
CLASSIFICATION	A machine learning task where the model assigns labels to inputs based on learned patterns.
CROSS-VALIDATION	A technique for assessing how well a machine learning model generalizes to new data by splitting the dataset into training and testing subsets multiple times.
DECISION TREE	A hierarchical model used in machine learning where decisions are made by following a set of branching conditions.
DISTANCE METRIC	A mathematical function that defines how the similarity or difference between two points is measured in a model, commonly used in k-NN.

EL	Ensemble Learning is a technique that combines multiple machine learning models to improve prediction accuracy, used in random forest.
GENERALIZATION	The ability of a machine learning model to perform well on unseen data, rather than just the training dataset.
GB	Gradient Boosting is a machine learning technique that improves predictive performance by training models sequentially.
GBM	Gradient Boosting Machines are an ensemble-based machine learning method that builds new models by iteratively combining weak predictors to reduce errors.
HYPERPARAMETER	A configuration setting in machine learning models that must be set before training, such as the number of neighbors in k-NN or the number of trees in random forest.
INTERPRETABILITY	The ability to understand and explain how a machine learning model makes predictions, an important aspect in policy applications.
IQR	The interquartile range is the spread of the middle 50% of a dataset, calculated as $Q_3 - Q_1$.
IQR OUTLIERS	An IQR outlier is any value below $Q_1 - 1.5 \cdot \mathrm{IQR}$ or above $Q_3 + 1.5 \cdot \mathrm{IQR}$. This rule highlights unusually distant points while remaining fairly robust to skewed data.
k-NN	k-Nearest Neighbors is a supervised learning algorithm that classifies or predicts values based on the nearest data points.

KERNEL FUNCTION	A mathematical function used in machine learning models like Support Vector Machines (SVM) to transform input data into higher dimensions.
LABEL	The target variable or outcome that a supervised learning model is trained to predict.
LIME	Local Interpretable Model-agnostic Explanations fits a simple surrogate model around a single prediction using perturbed samples and distance weighting, providing a fast, local approximation of how features influenced that specific output.
LOSS FUNCTION	A function that measures the error of a model's predictions compared to actual values, used to improve model accuracy.
MSE	Mean Squared Error is a common metric used to measure the accuracy of regression models by calculating the average squared differences between predicted and actual values.
NEET	NEET is an internationally used indicator for young people (typically aged 15–24) who are not in education, employment, or training.
NETFLIX PRIZE	The Netflix Prize was an open competition for the best collaborative filtering algorithm to predict user ratings for films, based on previous ratings without any other information about the users or films, i.e. without the users being identified except by numbers assigned for the contest. (Wikipedia Contributors 2019.)
NORMALIZATION	The process of scaling data to ensure all features contribute equally to model performance.

NON-LINEARITY	A characteristic of data where relationships between variables are not straight-line (linear), often requiring more complex models like random forest.
OVERFITTING	A problem in machine learning where a model learns patterns from the training data too well, resulting in poor performance on new data.
REGRESSION	A type of machine learning used for predicting continuous numerical values, such as unemployment rates.
REGULARIZATION	A technique used to prevent overfitting by adding a penalty to complex models.
RMSE	Root Mean Squared Error is a performance metric for regression models that measures the difference between predicted and actual values.
SCALING	The process of standardizing data so that different features contribute equally to the model.
SHAP	SHapley Additive exPlanations assigns each feature a fair, additive contribution to a model's prediction by approximating game-theoretic Shapley values, yielding locally accurate, model-agnostic explanations that can also be aggregated globally.
TEST SET	A portion of data used to evaluate the performance of a trained machine learning model.
TRAINING SET	A portion of data used to teach the machine learning model how to recognize patterns.
UNDERFITTING	A problem in machine learning where a model is too simple to capture important patterns in data, resulting in poor accuracy.

VALIDATION SET	A subset of data used to fine-tune machine learning models before final testing.
VI	Variable Importance is a measure used in random forest to determine which features contribute most to predictions.
WEIGHTING	A technique in machine learning where certain features or observations are given more importance than others.

1 INTRODUCTION

Addressing youth unemployment is a pressing concern worldwide, given its profound economic, social, and political ramifications. In Finland, the question is especially salient because local access to study options varies substantially between urban centres and sparsely populated regions, and for many young people study places are located far from home, which can make commuting or relocation financially difficult even when places are formally available.

The employment prospects of young people rely on the quality and accessibility of education, yet Finland displays notable regional differences in both educational provision and youth unemployment rates. This study introduces an inquiry into the effectiveness of education for early labour market entry in a place sensitive setting, not by restating attainment correlations but by examining whether variation in accessibility, understood as proximity to institutions and the breadth of local programmes, aligns with differences in youth unemployment across municipalities.

To examine these connections, national datasets from Statistics Finland, the Finnish National Board of Education and regional employment records are analysed with machine learning techniques that can detect non-linear associations and interactions. The models condition on factors such as distance to educational facilities, income levels and population density to assess patterns that conventional approaches might overlook, while remaining agnostic about causality at this stage. Machine learning is used because flexible nonparametric models can recover complex structure and latent or hidden interactions among covariates, revealing associations that simpler specifications and methods overlook.

Two further challenges frame the inquiry. First, if educational opportunities are distant, the direct and opportunity costs of travel, temporary housing and foregone earnings may limit participation for those with few resources. This raises the practical question of whether access is effectively realised. Second, after

education there remains the question of where jobs are located; the alignment between local programme mix and regional labour demand, including availability of work placements, may shape the transition from training to employment.

Through this lens, the study sets out a data driven agenda to describe how present day accessibility relates to youth unemployment and to identify where improved reach could plausibly matter most for early careers.

2 RESEARCH MOTIVATION AND OBJECTIVES

Across Finland, a young person's first step into working life is often decided long before a job interview, by whether the education they need is close enough, visible enough, and possible enough to pursue. In 2025, national unemployment is among the highest in the EU and volatility is pronounced for under-25s. The geography of opportunity has become harder to ignore, and travel costs and relocation constraints increasingly shape entry points to study and work. Many young people must commute long distances or move to larger cities to access upper secondary programmes, vocational training, workplace learning and first jobs. For those with fewer resources, the extra cost and logistics can turn nominally available pathways into practical dead ends, deepening regional disparities in education and employment opportunities (Reuters, 2025). Urban students usually have multiple study paths within reach, whereas students in remote municipalities face long travel times, limited programme choice, and fewer training opportunities and work placements. These patterns raise the question of how spatial and programme-related differences in educational accessibility relate to municipal-level youth unemployment in Finland, and how policy choices may further widen or narrow these gaps.

In this study, educational accessibility is understood as a combination of geographic proximity, breadth of local programme provision, and the time and money constraints that shape the real feasibility of participation. Proximity refers to the distance to relevant institutions, travel times, and the need for relocation; breadth of provision refers to the variety and volume of upper secondary and vocational education options, workplace learning opportunities and training placements available within a reasonable commute. Spatial accessibility interacts with the constraints that bind most tightly for lower-income youth and for those balancing family duties. In such circumstances, even small reductions in travel time or modest expansions in local programme choice can have outsized effects on participation. Digital and hybrid delivery can widen access only where reliable

connectivity, workplace training placements and local employer networks exist. This means that physical proximity, programme mix and local labour market ties will remain decisive for youth transitions into work, even under expanded online provision. These considerations motivate a georeferenced approach that captures both proximity to institutions and the breadth of local offerings, and that can be linked to municipal youth unemployment patterns.

Finnish research shows that higher educational attainment is associated with better employment outcomes and greater stability through downturns, including the 1990s recession and the 2008 financial crisis (Kalenius 2014; Sipilä, Kestilä & Martikainen 2011a, 121–131). Young people with only primary education face markedly higher risks of unemployment and long-term joblessness. Socio-economic disadvantages further heighten risks among the young, which means that barriers to access can entrench inequality if they coincide with weaker local provision (Sipilä et al. 2011b, 129). Persistent youth joblessness is linked to lower lifetime earnings, poorer health, and heightened social exclusion, and young men face a particularly high risk of exclusion from both education and work (Järvinen & Vanttaja 2005, 113–125; TEM 2009). These findings underline why it matters not only whether young people attain qualifications, but also how the spatial and institutional landscape either supports or obstructs their paths into further education and work.

European evidence indicates that recent vocational education and training graduates often achieve higher early employment than recent general education graduates, provided that vocational provision remains responsive to changing labour market demand (CEDEFOP 2017; CEDEFOP 2024). This pattern motivates attention to regional heterogeneity rather than average effects alone. At the same time, Finland is facing a planned 120 million euro reduction in vocational education funding that risks narrowing programme portfolios in smaller or rural municipalities and constraining capacity for work-based learning (OAJ 2024). When cuts interact with already thin programme menus and long travel distances, they may amplify disparities in both access and youth employment. CEDEFOP's evidence on the value of accessible, responsive vocational pathways for early career

outcomes and OAJ's warnings about the consequences of funding reductions together highlight the need to understand where improved accessibility can most plausibly reduce youth unemployment.

Against this background, the objectives of this study are threefold. First, it measures how present-day educational accessibility relates to municipal youth unemployment in Finland. Second, it identifies where and for whom improvements in accessibility are most strongly associated with lower youth unemployment. Third, it assesses how planned vocational funding cuts could modify these relationships.

The first objective is to link indicators of access, namely proximity to institutions and breadth of local programmes, with youth unemployment across Finland's 309 municipalities. This is done without re-establishing the known attainment-to-employment correlation that prior research has already documented (Kalenius 2014; Sipilä, Kestilä & Martikainen 2011a, 121–131). The second objective is to map regional and social heterogeneity using an analytical framework capable of detecting non-linearities and interactions, in order to reveal where marginal gains in access plausibly align with the largest reductions in youth unemployment and for which groups these gains are most pronounced. The third objective is to examine policy sensitivity under the planned reduction in vocational education funding, treating these cuts as contextual modifiers that may amplify disparities in both access and youth employment (OAJ 2024).

These objectives translate into three research questions:

1. To what extent is better geographic and programme access associated with lower municipal youth unemployment? In this question, access is measured as reduced distance to institutions and broader local programme offerings. Key socio-economic covariates are used to distinguish access-related patterns from background risks documented in prior studies (Kalenius 2014; Sipilä et al. 2011b, 129).

2. Where and for which groups are marginal improvements in accessibility most strongly aligned with reduced youth unemployment? In answering this question, the analysis takes non-linear responses and interaction effects into account. This makes it possible to identify municipalities and subpopulations for whom incremental reductions in distance or expansions in programme choice align with the largest reductions in unemployment.

3. How might vocational funding cuts shift these relationships across regions? In this study, they are treated as a policy shock that may attenuate or reverse beneficial access-to-employment patterns, particularly in municipalities already constrained in supply. In addition, how do these potential shifts appear in light of European evidence that labour-market-responsive vocational education supports smoother transitions from school to work (OAJ 2024; CEDEFOP 2017; CEDEFOP 2024)?

To address these questions, the study assembles municipal youth unemployment rates and georeferenced indicators of educational access, covering proximity to institutions and breadth of local offerings. It also includes socio-economic controls to mitigate confounding. A machine learning framework is used to capture non-linearities and interactions, to test associations, and to explore heterogeneity, without claiming causal effects. The planned vocational funding reductions are incorporated as contextual modifiers to assess how changes in provision could reshape access-to-employment relationships, particularly in municipalities where local provision is sparse or where young people face long and costly commutes to study (OAJ 2024). This design enables a flexible georeferenced analysis of municipal youth unemployment by combining educational accessibility indicators, socio-economic controls and vocational funding scenarios within a single non-linear modelling framework.

Taken together, the findings can guide targeted regional investment in programme provision, transport support, and employer partnerships, with priority to municipalities where marginal accessibility gains are most likely to reduce youth

unemployment. By clarifying how geographic and programme access relate to municipal youth unemployment under different policy scenarios, the study aims to inform decisions about where and how to sustain or expand provision so that education remains close enough, visible enough, and possible enough for young people across Finland to take their first steps into work.

3 THEORETICAL FRAMEWORK

Educational accessibility was examined for its association with municipal youth unemployment. The setting is complex, with non-linear relationships, interactions and heterogeneity that simple specifications may miss. In socio-economic data, relevant signals often lie in joint patterns among covariates rather than in single main effects, which weakens the leverage of traditional linear models. Machine learning (ML) is therefore warranted as it can adapt to high-dimensional structure and recover flexible response surfaces (Mullainathan & Spiess 2017, 87–106; Athey & Imbens 2019, 685–725). In this study, ML models are used as practical tools for detecting complex joint patterns in the municipal data that simpler linear specifications might miss.

Predictive models are not, by themselves, estimators of causal parameters; recent econometric work combines modern learners with identification strategies to make progress, for example through double or debiased machine learning for low-dimensional targets and through causal forests for heterogeneous effects under unconfoundedness (Chernozhukov et al. 2018, C1-C68; Wager & Athey 2018, 1228–1242). These strands disagree on emphasis. The policy-facing literature values accurate prediction for targeting and forecasting; the econometric literature insists on design and orthogonalization before interpretation. The present study aligns with both: flexible learners are used to map non-linearities and interactions, then results are read through designs and diagnostics that reduce confounding where possible.

Model transparency is addressed directly. Complex ensembles can obscure mechanisms, which is problematic in public policy. Post-hoc interpretability tools provide a partial remedy. SHAP values, for example, decompose a prediction into feature-level contributions with axiomatic guarantees. Local surrogate models such as LIME fit simple approximations in the neighbourhood of individual observations, yielding approximate explanations for complex models (Lundberg & Lee 2017; Ribeiro, Singh & Guestrin 2016, 1135–1144). These tools do not prove

causation, but they make fitted structure auditable, which matters when findings inform debates about regional provision and youth outcomes. In sum, machine learning is not adopted for novelty; it is used because the research question requires models that can detect latent interactions and context-specific responses, while recent advances allow the results to be interrogated, validated and, with appropriate designs, linked to causal claims.

Selecting the machine learning model for this study matters because it affects how well the models capture non-linear relationships between educational accessibility, socio-economic covariates and municipal youth unemployment. In practice, the choice must balance three needs: the ability to handle a relatively small but high-dimensional dataset, sufficient interpretability for municipal and national policy audiences, and enough flexibility in hyperparameter tuning to adapt to different model specifications. These criteria guide the comparison between random forest and k-NN in the present analysis.

ML models are well suited to situations where relationships are non-linear and context-specific, as is often the case when educational access, local labour markets and socio-economic risks interact (Kelleher, Mac Namee & D'Arcy 2015, 23). However, the usefulness of these models depends on the quality and representativeness of the underlying data, and estimates can be misleading if key assumptions are violated (Dangeti 2017, 40). In this thesis, ML predictions are therefore interpreted alongside Finnish institutional context and prior research, rather than treated as standalone evidence.

A vital innovation in machine learning is the rise of ensemble approaches, which combine multiple models to produce more robust predictions. Ensemble methods, such as bagging (Breiman 1996, 130) and random forests (Breiman 2001, 8), improve predictive stability by combining many slightly different trees and averaging their outputs. Drawing bootstrap samples and random subsets of features at each split reduces the influence of any single training sample or predictor and helps control overfitting (Ho 1998, 56; Chen & Ishwaran 2012, 125). In this study, this property is important because municipal-level data are relatively noisy

and contain interacting socio-economic and accessibility variables that a single tree might overfit.

Concerns about interpretability and transparency become more pressing when algorithms guide high-stakes decisions. Although decision trees present their logic in a relatively understandable flow (Quinlan 1986, 81), integrated ensembles often function more like black boxes (Friedman, Hastie & Tibshirani 2009, 90). This opacity challenges practitioners and policymakers who need to comprehend, explain, and justify recommendations. Researchers continue to develop techniques for model interpretation, such as feature importance measures and local surrogate models, so that complex predictions can still be grounded in human oversight (Raschka, Liu & Mirjalili 2022, 44). Full transparency is rarely attainable, but partial solutions may alleviate ethical and practical concerns, particularly in public policy domains where accountability is paramount.

Data integrity and the proper selection of metrics further influence the reliability of machine learning outcomes. Traditional algorithms based on nearest neighbor logic (Cover & Hart 1967, 3) rely heavily on distance metrics that may or may not capture real-world similarity. As Sharma (2024, 2) points out, choosing an ill-suited distance measure can distort classification boundaries and degrade performance. In addition, combining large datasets and a wide range of features can amplify biases if the datasets are not systematically vetted or if key variables are overlooked. The process of data preprocessing and feature engineering is therefore as critical to model success as the choice of algorithm itself (Witten, Frank, Hall & Pal 2016, 15). Thorough validation methods, including cross-validation and holdout tests, reduce the risk of inflated performance estimates and ensure that models generalize effectively to new data (Géron 2017, 45).

Budgetary considerations, for example the ongoing discussions about vocational education cuts (OAJ 2024), underscore the need for data-driven decisions that can balance cost efficiency with equitable societal outcomes. Machine learning can help uncover complex relationships in such contexts, for example by showing how resource distributions may influence employment trends for young people in

underserved areas. At the same time, policy decisions still depend on careful attention to data quality, interpretability and the limits of the models used, so that existing inequalities are not reinforced. In this thesis, ML predictions are therefore interpreted alongside Finnish institutional context and prior research, rather than treated as standalone evidence.

3.1 Selecting the machine learning model for this study

Random forest models frequently achieve top-tier accuracy and robustness, making them a favored choice in various predictive tasks. According to Caruana & Niculescu-Mizil (2006, 163), random forests often rank among the best-performing algorithms when tested against extensive benchmark datasets. This effectiveness can be attributed to their ensemble nature, as multiple decision trees are combined to reduce variance and guard against overfitting (Breiman 2001, 8). Furthermore, random forests tend to handle high-dimensional data with ease, partly because they sample both features and observations at each split, which promotes diversity among trees (Ho 1998, 56). Although they can sometimes lack transparency due to their complex structure (Louppe 2014, 75), random forests generally provide robust measures of variable importance, an advantage for projects requiring interpretability in policy or scientific contexts (Chen & Ishwaran 2012, 125).

k-Nearest Neighbors (k-NN) regression is a non-parametric approach that predicts a continuous target value by examining the outcomes of the k most similar instances in the training set. Originally popularized in the context of classification (Cover & Hart 1967, 3), the method has been extended to handle regression tasks by averaging, or otherwise combining, the numerical labels of the nearest neighbors. Because no explicit model is built in advance, k-NN regression relies heavily on the choice of distance metric and the local structure of the data (Friedman et al. 2009, 80). This local dependency can be beneficial in scenarios with erratic or highly variable relationships, as it does not force data to fit a predetermined functional form (Kleinberg & Tardos 2005, 62).

A key advantage of k-NN regression is its intuitive design and ease of implementation. When confronted with a new query point, the algorithm simply locates the *k* closest training samples, commonly determined through Euclidean or other specialized distance metrics. It then computes a prediction based on their average outcome (Witten et al. 2016, 19). Since there is no assumption of linearity or any particular distribution, the approach can adapt to highly non-linear patterns (Nilsson 1998, 15). This flexibility makes k-NN suitable for a range of real-world applications, from environmental modeling to financial forecasting. It is particularly useful when domain knowledge suggests that local neighbourhoods of data might better capture underlying relationships than a global, parametric model (Dangeti 2017, 50).

However, k-NN regression can suffer performance problems, especially when data become high-dimensional. The “curse of dimensionality” implies that distances in high-dimensional spaces can become less meaningful, thus making the nearest neighbor concept increasingly unreliable (Kleinberg & Tardos 2005, 79). Moreover, if the chosen distance metric does not reflect the true similarities in a given domain, the resulting predictions may be skewed (Sharma 2024, 2).

The findings and insights of Caruana & Niculescu-Mizil (2006, 161-168) are compiled into Table 1, which is ordered according to the suitability to this task. Random forest model especially has been proven to be highly suitable for analyses of socio-economic data. The best matches for the current problem are at the top of the table.

TABLE 1. The suitability of common ML models to socio-economic data (Caruana & Niculescu-Mizil, 2006, 161-168)

Method	Strengths	Weaknesses
Random Forest	High accuracy through ensemble averaging, robust to overfitting, handles high-dimensional data well.	Complex structure can hinder transparency and may require considerable computational resources for large datasets.
Gradient Boosting Machines	Strong predictive power via iterative error correction, can adapt effectively with careful regularization.	Potentially slow and resource-heavy to train, demands careful tuning of hyperparameters for best performance.
k-Nearest Neighbors	Simple to understand and implement, nonparametric, works well for small datasets.	Performance degrades with high-dimensional data, sensitive to noise and choice of distance metric.
Support Vector Machines	Effective in high-dimensional spaces, flexible kernel choices can capture non-linear relationships.	Choice of kernel parameters can be tricky, scaling to very large datasets can be computationally expensive.
Naïve Bayes	Fast training, works well with smaller datasets, relatively simple to interpret.	Strong independence assumptions can limit accuracy, may struggle with correlated features.

Random forest regression is the primary ML model for this study. A comparison will be made against k-NN, one of the oldest machine learning models dating back to 1960s (Cover & Hart 1967, 3), a long-time default choice for small datasets. Our dataset has less than 500 rows and less than 20 features, which makes it a small dataset (Witten et al. 2016, 55).

3.2 Random Forest Regression

Random-forest regression is an ensemble approach that aggregates many decision trees to enhance predictive accuracy and stability. It is commonly applied to

problems with non-linear relationships and high-dimensional feature sets, and it helps to control overfitting. By combining multiple decision trees, random forest regression enhances generalization and prevents individual trees from overfitting to the training data. (Breiman 2001, 5-7.)

The strength of random forest comes from Ensemble Learning (EL), a technique where multiple models, such as classifiers or regressors, are combined to produce a more accurate prediction than any single model alone. This principle is inspired by the wisdom of the crowd, where the collective judgment of a diverse group is often more reliable than that of a single expert. Similarly, in machine learning, ensemble methods improve performance by reducing variance, mitigating overfitting, and increasing robustness. (Géron 2017, 245.)

EL methods are widely applied in machine learning, particularly in the final stages of model development, where multiple well-performing models are combined to form a superior predictor. Many winning solutions in machine learning competitions, such as the Netflix Prize, have leveraged ensembles to achieve state-of-the-art performance. (Géron 2017, 245.)

Principles of decision trees and random forests

Decision trees and random forests are fundamental machine learning techniques widely used for classification and regression tasks. A decision tree is a hierarchical model that recursively partitions data into subsets based on feature values to make predictions. The tree consists of nodes where conditions are applied, branches that lead to further conditions, and terminal nodes, known as leaves, which contain the final output. Decision trees are effective in capturing complex relationships within the data, making them highly interpretable. However, they are prone to overfitting, particularly when the depth of the tree is unrestricted, leading to poor generalization on unseen data (Quinlan 1986, 81-106).

The random forest model was introduced by Breiman (2001) and is based on the principle of bagging, where multiple trees are trained on different random subsets of the data. Instead of relying on a single tree, the algorithm aggregates the

results of multiple trees to enhance generalization. The construction of a random forest begins with the creation of multiple training datasets using bootstrap sampling, a method where observations are randomly selected with replacement. (Breiman 1996, 123-140.)

Each decision tree within the random forest is constructed using a subset of features and data points. Unlike traditional decision trees, which evaluate all available features at every split, random forests introduce an additional layer of randomness by selecting a random subset of features at each node. If the total number of features in the dataset is denoted by p, then a commonly used heuristic is to consider only $\sqrt{p}$ features at each split (Ho 1998, 832-844.)

This ensures that individual trees do not rely too heavily on a single predictor, promoting diversity and reducing correlation among trees. Once all decision trees are trained, the final prediction is determined by aggregating their outputs. In regression tasks, the final prediction is obtained by averaging the outputs of all decision trees, which reduces variance and improves stability (Louppe 2014, 55-80.)

Mathematically, this process is expressed in Formula 1.

$$\hat{y} = \frac{1}{T}\sum_{t=1}^{T} \hat{y}_t \qquad \text{FORMULA 1}$$

T = total number of trees in ensemble
$\hat{y}_t$ = the predicted value from t-th tree
$\hat{y}$ = the final predicted value of the target variable

Hyperparameters and feature importance in RF Regression

Hyperparameter tuning is critically important in optimizing the performance of random forest regression. Model generalization is shaped by several hyperparameters, including the number of trees, the maximum depth permitted for each tree, the minimum sample size required to split a node, and the number of features evaluated at each split. Increasing the number of trees generally enhances

stability and reduces variance but also increases computational cost. A sufficiently large number of trees ensures convergence in performance, beyond which additional trees contribute minimally to accuracy (Breiman 2001, 6-32.)

The depth of each tree is another critical parameter. Deeper trees capture more complex patterns but are more prone to overfitting. Limiting the maximum depth prevents excessive branching, thereby maintaining generalizability. Another significant aspect of random forests is feature importance, which provides insights into how much each feature contributes to the model's predictions. The importance of a feature is typically measured based on the reduction in impurity it achieves across all trees. For regression tasks, impurity is quantified using the variance reduction criterion. At each node, the variance reduction achieved by a feature X_j when splitting into subsets R1 and R2 is given by Formula 2. (Friedman et al. 2009, 346.)

$$\Delta RSS = \sum_{i \in R_1} (y_i - \bar{y}_1)^2 + \sum_{i \in R_2} (y_i - \bar{y}_2)^2 \quad \text{FORMULA 2}$$

y_i = the actual value of the target variable
$\bar{y}_1, \bar{y}_2$ = the mean values of the target variable within subset R_1 and R_2
R_1, R_2 = subsets

Features that consistently produce greater variance reductions are ranked as more important. The means $\bar{y}_1$ and $\bar{y}_2$ are calculated with Formulas 3 and 4 respectively. (Friedman et al. 2009, 346.)

$$\bar{y}_1 = \frac{1}{|R_1|} \sum_{i \in R_1} y_i \quad \text{FORMULA 3}$$

$$\bar{y}_2 = \frac{1}{|R_2|} \sum_{i \in R_2} y_i \quad \text{FORMULA 4}$$

Random forests also allow for permutation-based feature importance estimation, where the values of a particular feature are randomly shuffled while keeping all other features unchanged. The resulting decrease in model performance indicates the significance of that feature. Features that, when permuted, lead to large increases in prediction error are considered crucial for the model, while those with minimal impact may be redundant. Feature importance rankings derived from random forests provide valuable insights into the most influential predictors, making the algorithm particularly useful in domains where interpretability is required. (Louppe 2014, 55-80.)

Practical applications of RF in regression tasks

Random forest regression has been widely applied across various fields due to its versatility, robustness, and high predictive accuracy. One of its most prominent applications is in economic forecasting, where it is used to predict unemployment rates, inflation trends, and economic growth indicators. By analyzing historical data and incorporating multiple socio-economic variables, random forests can identify complex relationships that traditional regression models might overlook. (Varian 2014, 3-28.)

In the domain of medical research, random forests have been utilized to predict disease progression, identify key risk factors, and improve diagnostic accuracy. The ability of random forests to rank feature importance allows healthcare professionals to determine which clinical indicators are most relevant in predicting patient outcomes. Studies have demonstrated that ensemble methods, including random forests, outperform conventional regression techniques in medical prognosis by reducing overfitting and handling heterogeneous data efficiently. (Chen & Ishwaran 2012, 323-329.)

Environmental science has also benefited from the application of random forest regression, particularly in climate modeling and ecological predictions. By incorporating geospatial data, climate variables, and historical trends, random forests have been employed to model the impact of climate change on biodiversity, forest cover dynamics, and pollution levels. The flexibility of random forests in handling

both numerical and categorical variables makes them well-suited for analyzing complex environmental datasets. (Prasad, Iverson & Liaw 2006, 181-199.)

These applications illustrate the adaptability of random forest regression in tackling real-world problems across multiple disciplines. Its ability to handle non-linearity, rank feature importance, and provide robust predictions has solidified its position as one of the most widely used ensemble learning techniques in modern machine learning. (Prasad et al. 2006, 181-199.)

The nearest neighbor algorithm is a widely used technique in machine learning and data analysis, particularly in classification, regression, and clustering tasks. It operates on the principle of proximity, making decisions based on the closest data points in a given feature space. In this study, the method provides a simple baseline model whose behaviour can be interpreted through the municipal neighbourhoods it uses for prediction.

The implementation of Random Forest Regression

The pseudocode of RF regression is presented in Figure 1. Each tree is trained on a different bootstrap sample of the original data, and at every split, only a random subset of the features is considered (Breiman 1996, 130; Breiman 2001, 8). This approach reduces the correlation among individual trees and helps control overfitting (Ho 1998, 56). Once all trees are constructed, new predictions are generated by averaging the outputs across the ensemble, a step which tends to stabilize the final estimate (Louppe 2014, 75).

```
Code begins.

Algorithm RandomForestRegression(D, B, m_try):
  // D is the training dataset with N samples
  // B is the number of trees in the ensemble
  // m_try is the number of features randomly selected at each split

  // Initialize an ensemble of trees, RF, to an empty set

  For b = 1 to B:
    1. Draw a bootstrap sample D_b of size N from D (sampling with replacement).
    2. Initialize a regression tree, T_b.
    3. Repeat until the tree is fully grown:
       a. Select m_try features at random from the total feature set.
```

```
      b. For the current node, choose the best split feature and split value
         from these m_try features (minimizing some impurity measure,
         e.g. MSE).
      c. Partition the node based on the chosen split.
   4. Add T_b to the ensemble RF.

  // For a new instance x:
  // 1. Predict with all trees in RF.
  // 2. Average the predictions to get the final output.

  Return RF

Code ends.
```

FIGURE 1. Random forest regression algorithm pseudocode

Appendix 1 shows a step-by-step example of the workings of random forest regression algorithm with two plots depicting the results. The whole process is run against a small 7 row dataset.

3.3 k-Nearest Neighbor algorithm

The k-nearest neighbours method is a fundamental predictive technique that imposes minimal assumptions about data structure. For a given query point, proximity is computed with a chosen distance metric and the k closest observations in feature space are identified. The algorithm predicts the target variable by averaging the values of these neighboring points. At the heart of nearest neighbor lies the concept of similarity or distance. (Friedman et al. 2009, 3-14.)

The nearest neighbor method differs greatly from linear models, which assume a roughly linear relationship between inputs and outputs and depend on global stability for predictions. While linear models can create stable and smooth decision boundaries, their inflexibility may result in errors when the data shows non-linear patterns. Nearest neighbor, however, is very flexible, creating irregular and locally responsive decision boundaries that fit well with localized data structures. This flexibility does come with a trade-off, as it can lead to higher variance and instability, particularly in small neighborhoods or noisy datasets. (Friedman et al. 2009, 3-14.)

The nearest neighbor algorithm functions by analyzing a set of labeled data points and determining the label of a new, unseen data point based on its similarity or closeness to the existing labeled data. The measure of closeness is typically determined by a distance metric, such as Euclidean distance, Manhattan distance, or cosine similarity.

According to Kelleher, Mac Namee & D'Arcy (2015, 231) "Similarity-based approaches to machine learning come from the idea that the best way to make a [sic] predictions is to simply look at what has worked well in the past and predict the same thing again. The fundamental concepts required to build a system based on this idea are feature spaces and measures of similarity."

The most straightforward way to assess the similarity of two instances, *a* and *b*, is to compute their distance in the feature space. We choose an appropriate distance metric to do this: *metric(a,b)* is a function that returns the distance between two instances *a* and *b*. (Kelleher et al. 2015, 238.)

In practical terms, this algorithm is particularly useful when there is no prior knowledge about the statistical properties of the data. It relies solely on the assumption that similar observations tend to have similar outcomes. To classify a new data point, the algorithm calculates distances to all previously observed points and assigns the new point to the category of its closest match. This decision-making process ensures a high degree of flexibility, allowing the method to adapt to various types of data without complex modeling. (Cover & Hart 1967, 21-27.)

The nearest neighbor approach has been widely studied and extended, including variants that consider multiple nearest neighbors to improve classification decisions. Its foundational idea of proximity-based classification has also inspired applications beyond classification, such as density estimation and clustering. While the algorithm is computationally intensive and sensitive to noise, its ability to closely approximate the optimal classification error in large datasets has cemented its importance in the field of pattern recognition and machine learning. (Cover & Hart 1967, 21-27.)

Variations and applications of k nearest neighbors (k-NN)

Numerous improvements have been made to overcome the limitations of the basic k-nearest neighbor (k-NN) method. For example, kernel methods substitute the binary weight given to neighbors with a continuous weighting system that decreases with distance, resulting in smoother predictions. Another variant, local regression, fits localized linear models instead of averaging constant values, providing more refined approximations. (Friedman et al. 2009, 3-14.)

The applications of k-NN cover both regression and classification tasks, where its simplicity and effectiveness make it especially suitable for low-dimensional data or situations that require minimal assumptions. It performs well in cases where the underlying distribution is complex, such as in mixture models with overlapping or distinct clusters. However, in high-dimensional spaces, the performance of k-NN declines due to the "curse of dimensionality," as the distances between points become less significant. (Friedman et al. 2009, 3-14.)

The selection of k is critical for the effectiveness of the algorithm. A *k* value as low as one would make the algorithm enormously susceptible for noise impact, as it would consider only the nearest neighbor, which could possibly be an outlier. Conversely, a very high value of *k* reduces the sensitivity of the procedure to the local geometry of the data. It does so by relying on information from a larger number of possibly distant points that might not have a direct relationship with the query. Thus, *k* creates a balance between capturing a localized detail and generalization, and determines the application area with a likely dependability on the data and the solved question. (Raschka et al. 2022, 53-59.)

Distance metrics in Machine Learning

Distance metrics are crucial in machine learning, particularly in algorithms that depend on the similarity between data points. They offer a measurable way to assess how close or distant two points are within a specific feature space, which is vital for various tasks such as classification, clustering, regression, and anomaly detection. By evaluating the relationships between data points, distance

metrics significantly impact the decisions and accuracy of many algorithms, including k-Nearest Neighbors (k-NN), support vector machines (SVM), and clustering techniques like k-means. (Sharma 2024.)

A distance metric is a mathematical function that quantifies the "distance" between two points in a dataset. These points are typically represented as vectors in a multi-dimensional feature space, where each dimension corresponds to a specific feature or attribute. The selection of a distance metric is influenced by the characteristics of the data and the problem at hand. (Sharma 2024.)

The choice of distance metric plays a crucial role in the effectiveness of k-NN and other machine learning algorithms that depend on proximity. While Euclidean distance is the most widely used metric due to its simplicity, it assumes that all dimensions are equally important, which may not always be the case. Other options like Manhattan or Minkowski distances, offer greater flexibility in how dimensions are weighted. Minkowski distance, in particular, provides a family of metrics that can be adjusted based on specific needs (Friedman et al. 2009, 3-14).

In high-dimensional spaces, the effectiveness of standard distance metrics can decrease because distances tend to concentrate, making all points seem equally distant from one another. This challenge calls for adjustments in how distances are calculated, such as normalizing dimensions or using metrics tailored to the domain that consider the varying significance of different features. More advanced methods involve learned metrics, where the significance of dimensions is derived from the data itself, or utilize domain-specific similarity measures, like cosine similarity for text data or Mahalanobis distance for datasets with correlated features. These customized metrics improve the performance of k-NN and similar algorithms, helping them stay effective and reliable across a range of different problem scenarios. (Friedman et al. 2009, 3-14.)

The importance of distance metrics

The choice of distance metric directly impacts the performance and behavior of machine learning algorithms. Instance-based learning postpones the learning

process until the classification stage, which is different from eager learning that generalizes data right away. It uses distance metrics to compare new instances with those already stored, employing methods like nearest neighbor or k-nearest-neighbor classification to determine class assignments. These distance metrics can effectively manage numeric attributes through techniques such as Euclidean distance, and they can also be adapted for nominal attributes using binary distinctions or more sophisticated measures. The importance of attribute weighting is crucial in emphasizing significant features, while efficient storage and selection of training instances are vital to minimize computational costs and ensure effective classification. (Witten, Frank, Hall & Pal 2016, 85-88.)

Instance-based learning is heavily dependent on distance metrics for representing implicit knowledge, creating boundaries in the instance space to differentiate between classes. Although this approach may not provide explicit generalization, it establishes decision boundaries through the chosen instances, enhancing computational efficiency by eliminating redundant data. More advanced techniques, such as rectangular generalizations, further improve classifications by dividing the instance space into overlapping or nested regions, allowing for exceptions to general rules. These modifications lead to more accurate classifications and tackle problems like overlapping rule applicability that are common in other learning methods. (Witten et al. 2016, 85-88.)

Distance metrics also play a significant role in clustering, which is another aspect of instance-based learning. Clusters group instances based on their similarities, laying the groundwork for further structural analysis, such as creating decision trees or rule sets. Clustering algorithms can utilize probabilistic, categorical, or hierarchical methods to organize data, often represented through visual tools like dendrograms or Venn diagrams. By emphasizing relationships and categorizing data into meaningful groups, clustering enables deeper insights and serves as a basis for more intricate machine learning tasks. (Witten et al. 2016, 85-88.)

Minkowski distance

The Minkowski metric plays a significant role in machine learning and AI. Many well-known machine learning algorithms utilize specific distance metrics like this one to assess the similarity between two data points. The Minkowski metric is particularly effective for numerical datasets when you need to evaluate the similarity in size among multiple data point vectors. (Raschka et al. 2022, 101-102.)

In k-NN, the Minkowski distance is a general family that includes Euclidean and Manhattan as special cases; it measures separation in a multi-dimensional space by summing the absolute differences of their coordinates raised to a specified power p, and then taking the p-th root of this sum. The parameter p determines the nature of the distance calculation, allowing the metric to adapt to different data and problem contexts. When $p = 2$, Minkowski distance reduces to Euclidean distance, emphasizing straight-line proximity, while $p = 1$ yields Manhattan distance, which measures distance based on axis-aligned paths. Minkowski distance is calculated according to Formula 5. (Şuhubi 2003, 271-274.)

$$d(x, y) = \left(\sum_{i=1}^{n} (x_i - y_i)^2 \right)^{1/p} \qquad \text{FORMULA 5}$$

d = distance

x, y = datapoint coordinates

i = index

p = an integer defining the type of distance

This flexibility makes Minkowski distance a powerful tool for applications where different types of feature relationships need to be captured. By adjusting p, users can control how much emphasis is placed on larger or smaller differences across dimensions. For instance, higher values of p make the metric more sensitive to large variations in certain features, as these dominate the sum more strongly. This adaptability allows the k-NN algorithm to better align with the specific characteristics of the dataset, enhancing its predictive performance. (Raschka et al. 2022, 101.)

However, the choice of p significantly influences the algorithm's results, and its appropriateness depends on the nature of the data. For example, datasets with diverse feature ranges may require careful preprocessing to avoid skewing the distance calculations, particularly for higher p values. Minkowski distance, like other metrics, is also sensitive to high-dimensional data, where distances tend to lose meaningful variation. Despite these challenges, its versatility makes it a widely applicable and effective distance measure in the k-NN algorithm. (Raschka et al. 2022, 101-102.)

Manhattan distance

Manhattan distance, often referred to as taxicab distance or L_1-norm, is a metric utilized in the k-NN algorithm to determine the distance between two points in a multidimensional space. It calculates the total absolute difference between the coordinates of the two points across all dimensions. Unlike Euclidean distance, which measures the shortest straight-line distance, Manhattan distance only considers horizontal and vertical movements, similar to navigating a grid-like city layout. This feature makes it especially useful for data where the relationships between features are discrete or grid-like. The Manhattan distance is calculated according to Formula 6. (Şuhubi 2003, 271-274.)

$$d(x, y) = \sum_{i=1}^{n} |x_i - y_i| \qquad \text{FORMULA 6}$$

d = distance
x, y = datapoint coordinates
i = index

In the context of k-NN, Manhattan distance is frequently favored when the relationships between data points are not effectively represented by straight-line proximity. It performs well with high-dimensional data, where the Euclidean metric can be skewed by larger feature scales. Moreover, since it avoids the computational burden of squaring and square-rooting differences, Manhattan distance can be more efficient in certain scenarios. (Raschka et al. 2022, 101.)

This metric is sensitive to how features are scaled, so it's crucial to normalize or standardize the data to ensure that each feature contributes equally to the distance calculation. Its straightforward nature and intuitive interpretation make it a popular choice in problems involving ordinal data or where axis-aligned relationships are significant. By examining individual dimensions independently, Manhattan distance offers a different perspective on similarity that complements other distance measures in k-NN applications. (Kelleher et al. 2015, 239-240.)

Euclidean distance

Euclidean distance is a method through which the distance or straight distance between any two points in multi-dimensional space can be measured. It determines the entire dissimilarity of a query point with a data point through evaluating the square root of the addition of all squared differences among all the dimensions. Each dimension can denote a feature, and the difference in every dimension is squared to create an entirely positive summed value. This is a method that works particularly well with continuous numerical features, with comparison associated with a distance in geometry translating well into similarity in data. Euclidean distance is calculated according to Formula 7. (Nilsson 1998, 71.)

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$ FORMULA 7

d = distance
x, y = datapoint coordinates
i = index

Euclidean distance is sensitive to the scale of the features, as a very large feature can drastically change the distance computed. Therefore, it is mostly normalised or standardised before applying the metric. Equal contribution of all features in computing distance makes it reliable when it comes to measuring a true similarity; and, therefore, it is typically used in k-NN because it is simple and

matches intuitive feelings regarding proximity, making it, thus, a natural choice for numerical data in lower-dimensional problems. (Nilsson 1998, 71-73.)

Cosine similarity

Cosine similarity is a metric used in the k-NN algorithm to assess how similar two vectors are by looking at the angle between them instead of their size. It computes the cosine of the angle formed by the vectors in a multidimensional space, resulting in a value that ranges from -1 to 1. A cosine similarity of 1 means the vectors are perfectly aligned, 0 indicates they are orthogonal (showing no similarity), and -1 signifies they are directly opposite each other. This metric is especially useful for datasets where the direction of the data points is more significant than their size, such as in applications involving text or document similarity. The cosine similarity is calculated according to Formula 8. (Kelleher et al. 2015, 218-220.)

$$S_C(A,B) = \frac{\boldsymbol{A}\cdot\boldsymbol{B}}{\|\boldsymbol{A}\|\|\boldsymbol{B}\|} = \frac{\sum_{i=1}^{n}\boldsymbol{A}_i\boldsymbol{B}_i}{\sqrt{\sum_{i=1}^{n}\boldsymbol{A}_i^2}\cdot\sqrt{\sum_{i=1}^{n}\boldsymbol{B}_i^2}}$$ FORMULA 8

S_C = cosine similarity
$\boldsymbol{A}, \boldsymbol{B}$ = n-dimensional vectors
$\boldsymbol{A}_i, \boldsymbol{B}_i$ = ith components of vectors A and B
(Şuhubi 2003, 500-511.)

In k-NN, cosine similarity is frequently applied when the data consists of high-dimensional sparse vectors, like word embeddings or term-frequency vectors in natural language processing. By concentrating on the angular relationship, it reduces the impact of variations in vector size that can arise from feature scaling or normalization. This makes cosine similarity an effective metric for identifying relationships in datasets where the relative proportions or patterns in feature values are more meaningful than absolute differences. (Kelleher et al. 2015, 218-220.)

Unlike distance metrics such as Euclidean or Manhattan, cosine similarity does not take into account the actual spatial distance between points but focuses on their orientation. This difference makes it a complementary option for problems where directional alignment is more crucial than physical closeness. While it works well in many situations, cosine similarity may not be as effective when differences in magnitude are important to the task. Still, its capacity to reveal relational structures in data makes it a valuable tool in the k-NN algorithm for specific applications. (Dangeti 2017, 280-282.)

Practical implementation of k-NN

The pseudocode for the k-nearest neighbor algorithm is shown in Figure 2. The k-nearest neighbors are found by sorting all samples according to their distance from the query point (Cover & Hart 1967, 3). The predicted value is then computed by taking an average (or another aggregated statistic) of the labels corresponding to the nearest neighbors (Friedman et al. 2009, 80). While Euclidean distance is common, domain-specific metrics may yield better results, and thus the choice of distance metric is an integral consideration (Witten et al. 2016, 19).

```
Code begins.

Algorithm KNearestNeighborRegression(D, k, distance_metric):
  // D is the training dataset consisting of N samples { (x_i, y_i) }
  // Each x_i is a feature vector, y_i is a continuous label
  // k is the number of nearest neighbors to consider
  // distance_metric is a function to compute distances between data points

  For a new query point x*:
    1. Create an array Distances to store (distance, label) pairs.
    2. For each sample (x_i, y_i) in D:
       a. Compute dist = distance_metric(x*, x_i)
       b. Append (dist, y_i) to Distances
    3. Sort Distances by ascending distance.
    4. Take the first k entries from Distances, i.e., the k nearest neighbors.
    5. Compute prediction = average of the y_i values for those k neighbors.
    6. Return prediction as the model's output.

Code ends.
```

FIGURE 2. k-Nearest Neighbor algorithm pseudocode

Appendix 2 shows a step-by-step example of the workings of k-NN algorithm with a plot depicting the end result. The whole process is run against a small 7 row dataset.

4 METHODS

4.1 Data Collection

The analytical dataset was assembled from three complementary sources to capture, at municipal resolution, educational supply, population size, territorial context, and youth unemployment. Figure 3 shows the data collection, processing and merging pipeline.

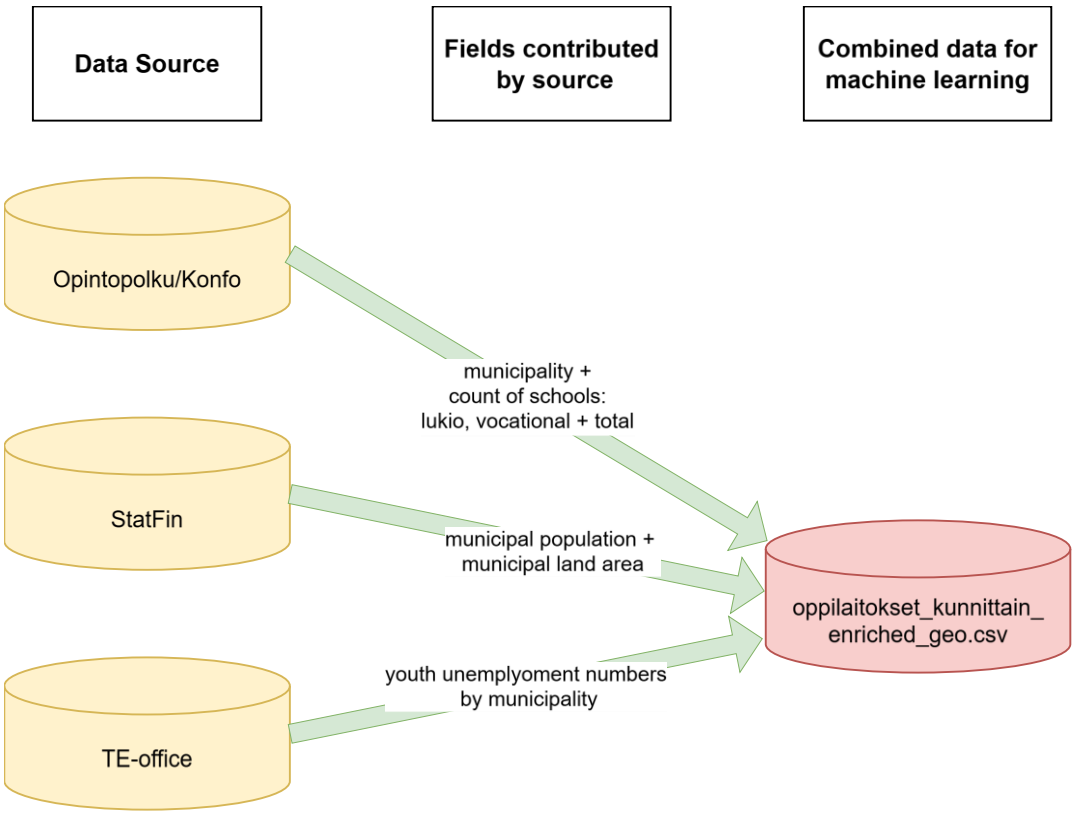

FIGURE 3. The data collection pipeline

First a list of upper-secondary institutions was retrieved for every Finnish municipality from Opintopolku's public Konfo service. The service returns the results as JSON; each item describes an institution, its provider, and any listed locations. The records were then cleaned, obvious duplicates were removed, and each

institution was placed into one of three groups: general upper secondary (lukio), vocational, or other. The summary was delivered as a cleaned CSV file suitable for integration. The python code that was used to download and preprocess this data is in Appendix 3.

Second, youth-unemployment counts by municipality were supplied from the TE administration. The file is loaded from https://temtyonvalitystilasto.fi according to the selection parameters shown in Figure 4. The downloaded data has an awkward layout and requires light processing. The code to process the data is in Appendix 4. The file contains annual series with “Sukupuolet yhteensä” (both sexes). The latest observation per municipality was extracted and normalised by population to produce a rate per 10,000 residents, which serves as the primary response variable.

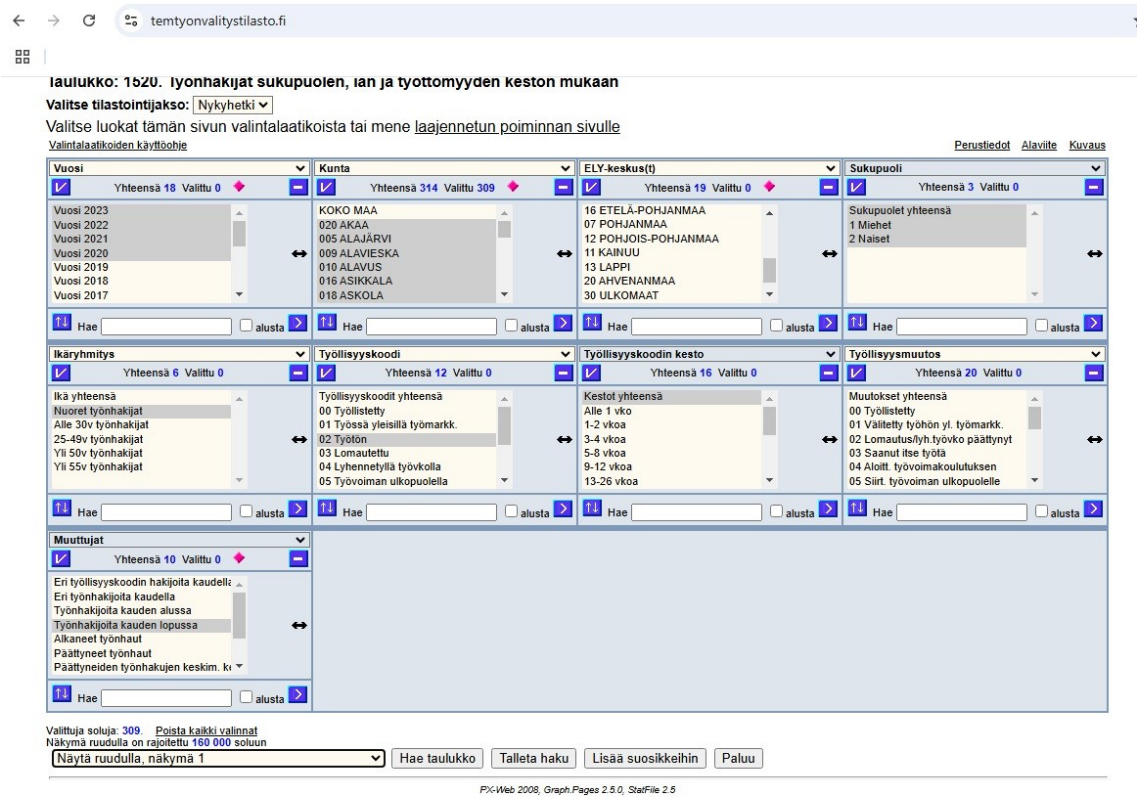

FIGURE 4. The TE-office database query parameter selection and download page.

Lastly, municipal population and land-area data were retrieved from Statistics Finland (StatFin) via the PxWeb API: https://pxdata.stat.fi/PXWeb/api/v1/fi/Kuntien_avainluvut/Kuntien_avainluvut__2021/laaja_alueaikasarjat_2021.px

These figures provide the headcounts needed to construct rates per 10,000 residents and, together with land area, to compute population density.

The JSON response was parsed into a Pandas Dataframe by expanding the dimension categories and mapping each observation to its descriptive labels. The table was then pivoted to a wide format so that population, land area, and (where available) total area appear as separate columns. These geographic variables were standardized to canonical names and joined to the institutions table using a normalized municipality key. Finally, the Opintopolku/Konfo and TE Office datasets were merged at the municipal level, and the integrated dataset was saved as oppilaitokset_kunnittain_enriched_geo.csv for subsequent machine-learning analysis. Appendix 5 lists the corresponding code.

The integrated dataset therefore combines: (i) institutional counts by type and in total, (ii) municipal population totals for the latest year, (iii) municipal land and total area, and (iv) the latest youth-unemployment counts. All sources are official or derived directly from official outputs to maximise coverage, comparability, and reproducibility. Reliance on StatFin and MML aligns administrative boundaries and reference dates; the TE series provides the labour-market outcome in a compatible municipal scheme. This combination enables derivative indicators such as population density and institutions per 10,000 residents, facilitating scale-aware comparisons across municipalities of different sizes and settlement structures. Separation of institutional types supports exploratory assessment of whether segments of the education supply are differentially associated with youth unemployment.

4.2 Data preprocessing

Exploratory data analysis is conducted to understand the empirical structure of the dataset before formal modelling begins. Its purpose is to assess data quality, reveal distributions and ranges, detect missingness and outliers, and surface preliminary relationships among variables.

Assumptions implicit in later modelling steps are stress-tested here; violations can be identified early and addressed through cleaning, transformation, or feature engineering. Insights from EDA guide sensible choices of target definition, candidate predictors, and evaluation strategy, while helping to avoid leakage and spurious correlations.

In a typical workflow, EDA follows data acquisition and preprocessing, and it precedes model selection and validation. By clarifying what the data can and cannot support, EDA reduces avoidable iteration and improves the credibility of subsequent inference.

Before modelling, the datasets were aligned to a common municipal key so that records could be joined reliably. Core indicators were then derived to make municipalities comparable. Population density was computed from land area, institutions per 10,000 residents captured supply relative to scale, and the target was defined as youth unemployed per 10,000 residents. Basic completeness checks were performed, and observations that could create invalid rates were flagged. This preprocessing was undertaken to reduce noise, improve comparability, and ensure that subsequent analyses reflect substantive patterns rather than formatting artefacts. The code to conduct the cleanup and EDA is in Appendix 6.

Exploratory data analysis was conducted after the initial data cleanup to understand structure and plausibility before modelling. Outliers were screened with an interquartile range rule for the target rate, youth unemployment counts, total institutions, institutions per 10 000 residents, and population density. Figure 5 shows the share of outliers by metric; small municipalities were flagged often for rate measures because small denominators amplify variability, whereas large

cities appeared as high leverage for population and density. Flags were retained for transparency, since the aim was exploration.

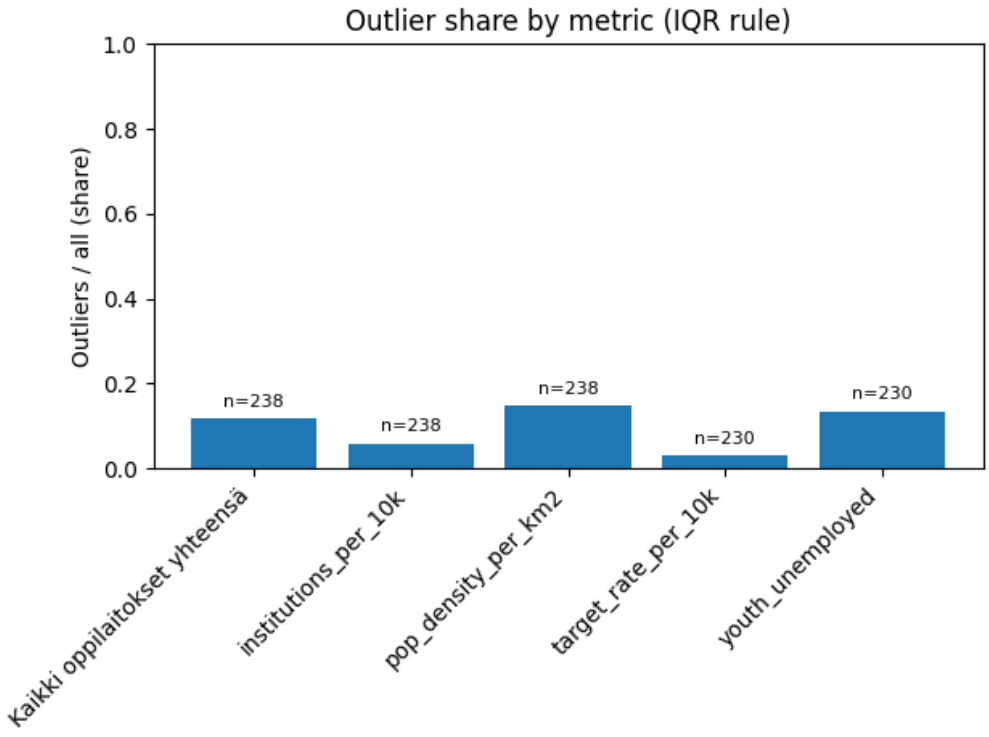

FIGURE 5. Outlier share by metric

Flags were retained for transparency and not removed automatically, since the objective was exploratory. Sensitivity analyses can, however, benefit from trimming or robust estimators.

A Pearson correlation heatmap was produced for the main variables (Figure 6). Strong associations were observed among population, total institutions, and population density, reflecting the co-movement of scale, concentration, and service availability. By contrast, the target rate showed only modest linear correlation with education supply once population scaling was considered. The per capita supply measure reduced mechanical links to population, but it did not yield a strong linear association with the target. Together, these results suggest that additional covariates and non-linear relationships are likely. On this basis, the prepared dataset was considered fit for modelling.

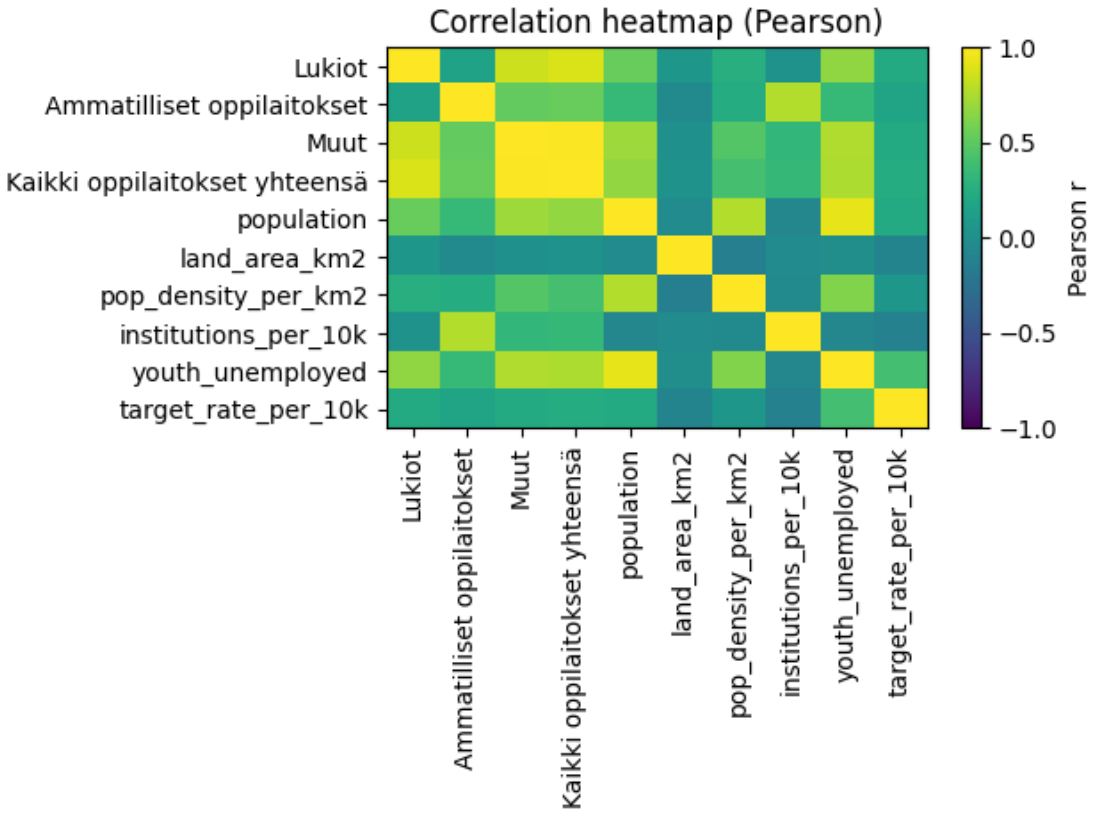

FIGURE 6. Correlation heatmap (Pearson) of features.

4.3 Machine Learning models

k-Nearest-Neighbours

A k-nearest neighbours (k-NN) regression model was adopted as the initial predictive approach to provide a transparent baseline with minimal structural assumptions. Because the method is non-parametric, no global functional form is imposed; local structure in the data is exploited, which is suitable when heterogeneous municipal contexts and latent non-linearities are plausible. Case-level interpretability is retained, since each prediction can be linked to a neighbourhood of similar municipalities, enabling proximity-based diagnostics. The code to implement the k-NN regression is shown in Appendix 7.

Model behaviour was governed by a small set of controls. Sensitivity to feature scale was mitigated through standardisation prior to distance computation. Neighbourhood size regulated the bias–variance trade-off, and distance weighting assigned greater influence to nearer observations, improving stability when neighbourhood composition varied. Computational requirements remained modest for the dataset size. Because predictions are anchored in observed exemplars, fitted values stayed within empirically realistic ranges.

Random forest regressor

A random forest (RF) regression model was selected as the principal estimator to accommodate non-linear relationships, interactions among predictors, and heteroscedastic behaviour across municipalities. Variance was reduced by aggregating many decorrelated decision trees, which limited overfitting while preserving flexibility to learn complex response surfaces without specifying a functional form in advance. Invariance to monotone rescaling and tolerance for mixed feature magnitudes further supported this choice.

No explicit distance metric was required. Similarity was induced implicitly through recursive partitioning, with splits chosen to maximise reductions in squared error. The bias–variance profile was shaped by structural parameters, including the number of trees, maximum depth, minimum node sizes, feature subsampling, and bootstrap. Together, these controls allowed rich interaction capture while constraining memorisation in small samples. Post-hoc variable-importance profiles provided an interpretable window into learned structure.

4.4 Evaluation and Validation

Model performance was measured using root mean squared error (RMSE) and the coefficient of determination (R^2). RMSE, computed as the square root of the mean of squared residuals, quantifies average prediction error in the natural units of the target (youth unemployed per 10,000 residents), facilitating direct substantive interpretation. R^2 summarises the share of variance in the response

explained by the model relative to a mean-only baseline; values closer to 1 indicate stronger explanatory power, while values near or below 0 signal little or no improvement over the baseline.

Validation proceeded in two complementary stages. First, a held-out test split was created to provide an unbiased estimate of generalisation performance after model development. All modelling choices were finalised without reference to this partition, and the final metrics were reported on the untouched test data. Second, k-fold cross-validation was used within the training portion to stabilise model selection and hyperparameter tuning. The training data were partitioned into k folds; models were fit on $k-1$ folds and evaluated on the remaining fold, rotating across all folds, and the resulting R^2 scores were averaged to reduce sampling variance.

For the RF regressor, a randomised hyperparameter search over plausible ranges (e.g., number of trees, maximum depth, minimum samples per split/leaf, feature subsampling, bootstrap) was embedded inside cross-validation, with mean CV R^2 serving as the selection criterion. The best configuration was then refit on the entire training set and finally assessed on the held-out test set using RMSE and R^2. For k-NN, neighbourhood size and distance weighting were analogously chosen using cross-validated performance on the training folds, after feature standardization. This protocol ensures that reported results reflect out-of-sample behaviour and that model complexity is controlled through data-driven, reproducible procedures.

5 RESULTS

5.1 Predictions and performance metrics

Predictions targeted municipal youth unemployment per 10,000 residents. They used a shared feature set: *Lukiot, Ammatilliset oppilaitokset, Muut, Kaikki oppilaitokset yhteensä, population, pop_density_per_km2,* and *institutions_per_10k*. Models were tuned by cross-validation on the training fold and evaluated on a held-out test fold.

The plot (Figure 5) compares k-NN's test predictions to actual values, and the wider spread around the identity/best-fit line reflects greater variance and weaker alignment of neighbourhood-based predictions with the true rates.

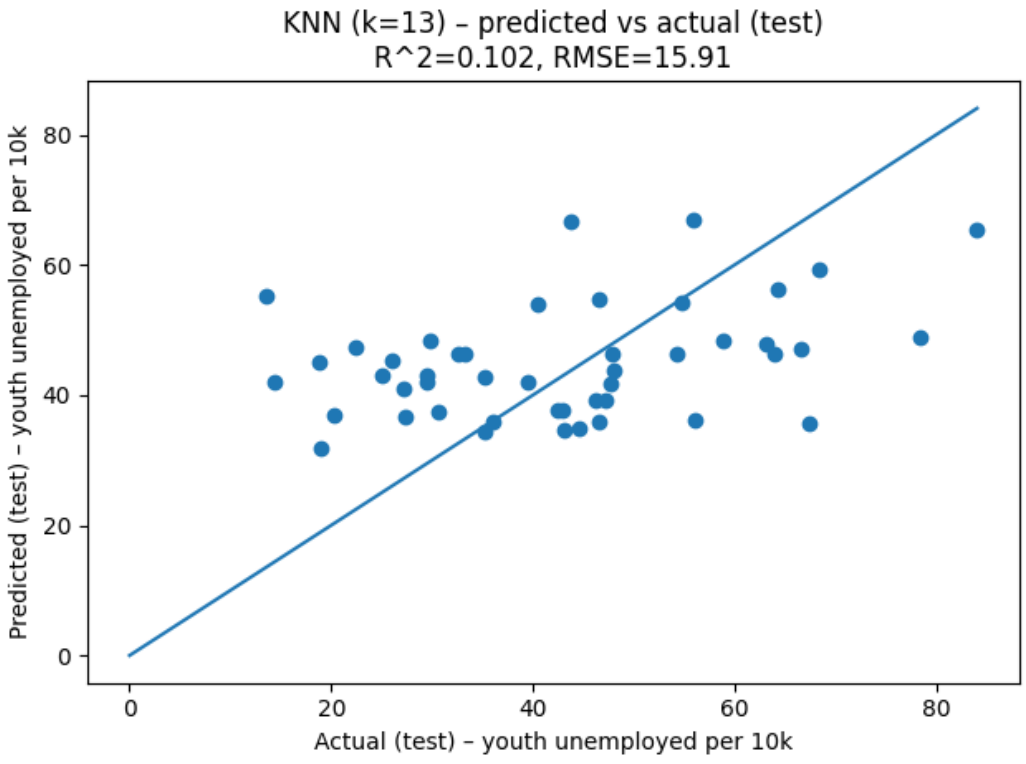

FIGURE 5: Scatterplot of k-NN predicted vs actual, with best fit line

The predictive power of the k-NN model proves to be quite weak. Part of the cross-municipality variation in youth unemployment rates is predictable from the supplied predictors, yet substantial residual variance remains. The k-NN baseline captured limited structure: predicted–actual scatterplots exhibited a positive slope with wide dispersion around the identity line, consistent with sensitivity to noisy rate denominators in small municipalities and the method's reliance on local averaging.

The random forest exhibited tighter alignment with the identity line, indicating improved calibration and lower error. Figure 6 shows the scatter plot. With respect to the relationship of interest, both models learned a weak negative association between institutions per 10,000 residents and youth-unemployment rates after accounting for population scale and land-area density. The effect remained small relative to the influence of overall population and density.

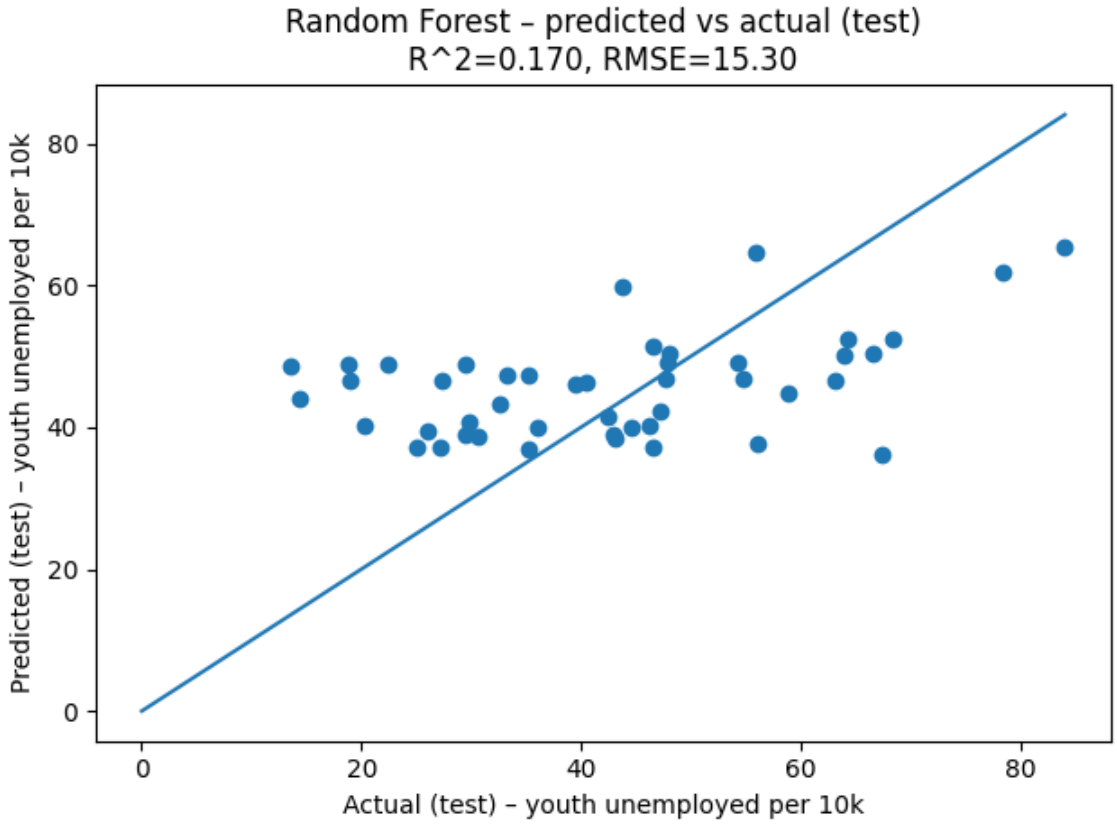

FIGURE 6. Scatterplot of RF predicted vs actual, with best fit line.

5.2 Comparing the selected ML models

The two estimators provide complementary perspectives. k-NN offers locality and case-level transparency, which aids qualitative checks but suffers when neighbourhood geometry is distorted by heteroscedastic noise; a single capacity parameter limits representation of higher-order interactions. RF aggregates many decorrelated trees, capturing non-linear effects and interactions without prespecifying functional forms.

Feature-importance diagnostics consistently emphasised population and pop_density_per_km2, with total institutions contributing next and institutions per 10,000 residents adding a secondary, directionally negative signal. Taken together, the models indicate that denser education networks are associated with slightly lower youth unemployment, but the association is modest once municipal scale and density are considered. Residual error magnitudes suggest that some determinants are omitted. These include local labour demand, socio-economic composition, programme mix and capacity, and travel-time accessibility, all of which are needed for higher precision.

6 DISCUSSION AND CONCLUSION

6.1 Interpretation of results

The machine learning analysis does not confirm the initial expectation that educational accessibility would show a strong imprint on municipal youth unemployment. After conditioning on observable covariates and allowing for sampling variability, the associations between provider density and unemployment remain modest. This section interprets why the signal is weak and how that aligns with previous research.

A modest signal is plausible given the way accessibility is measured. Provider counts and coarse demographics are blunt proxies for the mechanisms that actually shape youth outcomes. Youth unemployment reflects an equilibrium in which individual endowments, household resources and local labour demand interact. Marginal variation in upper secondary provision, measured only as institutional density, does not dominate this equilibrium. Finnish NEET (not in education, employment, or training) evidence indicates that municipal differences are driven mainly by individual factors and area demand. The measured availability of public services shows no direct association with NEET rates once other factors are held constant. This helps explain why institutional density in this study is only weakly linked to youth unemployment (Ahlroth, Laaninen and Niemelä 2025).

Sampling volatility in small municipalities further weakens the fit. Small absolute changes in unemployed youth generate large swings in per-capita rates, which adds noise to the outcome variable. Allowing for non-linearities and interactions improves fit somewhat. This is consistent with access mattering in specific contexts. For example, access may matter more when programme breadth aligns with settlement structure and entry-level demand, but it still does not produce large average effects.

Social frictions linked to leaving home impose additional constraints that simple access measures do not capture. Educational mobility may weaken peer networks and increase loneliness around transitions. European evidence also points to mental health strain during such moves (Sundqvist et al. 2024, 1–23; Guerrero-Puerta & Guerrero 2023, 103). Finnish work connects residential mobility and school changes with risks to social inclusion, which implies that distance-induced mobility and relocation impose hidden costs (McMullin 2021, 16–27; University of Turku 2025). These costs likely interact with household resources and with the availability of local work placements. The same nominal institutional access can therefore yield very different labour market consequences across places and groups.

Economic frictions operate in parallel. Longer commutes reduce study time and academic performance in quasi-experimental settings, which indicates a time-budget mechanism that institutional counts do not capture (Serra, Freitas and Balcão Reis 2021). Slower progression or non-completion then mutes downstream employment effects in commuter regions. On the demand side, persistent occupational and geographical mismatch weakens the mapping from local education supply to local youth jobs. When programme mix and vacancies are misaligned, more providers per resident do not translate into clearly lower youth unemployment (Uusitalo et al. 2024; Jauhiainen 2011, 338–345). This helps rationalise why large education cities can display high youth unemployment despite dense provision. Inflows of students and graduates may outpace short-run demand and concentrate search frictions. Internal migration may reduce slack in other regions yet induce selection that biases municipal correlations downward (Kultalahti 2001, 3; Tuononen 2019).

Student support can moderate some of these frictions, but evidence is mixed. Grants, loans and housing benefits administered by Kela reduce liquidity constraints and support independent living, which should ease access for those whose study places are distant. Administrative assessments report improved real incomes among aid recipients, which is consistent with partial mitigation of cost barriers (National Audit Office of Finland 2022; Kela 2025a; Kela 2025b). At the

same time, work during studies and income-limit rules have behavioural effects. Reforms show only modest or field-specific impacts on time to degree. Rising reliance on loans and recovery of overpaid aid suggest that affordability remains a moving target. This weakens any expectation that access alone would yield large municipal-level reductions in youth unemployment (Häkkinen 2002; Kylliäinen 2024; Hietala 2025; Yle 2025).

Education remains an effective channel for labour market entry, but the channel is sensitive to programme–demand matching and to the quality of school-to-work bridges. European comparisons show higher employment rates for recent vocational education and training graduates than for recent general education graduates. This supports the premise that well matched provision improves early employment (CEDEFOP 2024; CEDEFOP 2025). The advantage depends on responsiveness to local demand and on employer linkages such as internships and thesis placements. These linkages remain uneven across regions and student groups (E2 Research 2023). These studies suggest that the feature set in this thesis could be strengthened. Useful additions would include travel time to the nearest relevant programme, measures of placement or apprenticeship availability, vacancy data by field, and indicators of household disadvantage. Without such refinements, institutional counts and broad demographics will continue to provide only limited predictive leverage.

Taken together, the weak dependence between provider density and youth unemployment is consistent with a system in which access is necessary but not sufficient. Realised benefits are mediated by the social and economic frictions of leaving home, by commuting and costs, and by the alignment of programme mix with local demand. Evidence points both to the promise of accessible, well matched provision and to the limits of access as a stand-alone lever. This clarifies why the models detect structure yet achieve only modest out-of-sample fit.

6.2 Strengths and weaknesses ML models

The two estimators used in this study capture the same underlying signal and lead to similar conclusions. This consistency indicates that the machine learning framework is appropriate for the task and that the weak dependence between school density and youth unemployment is a genuine feature of the data, not an artefact of model choice.

The k-nearest-neighbours regressor provides locality and case level transparency. Its predictions are based on outcomes for municipalities with similar characteristics, which supports plausibility checks and communication with practitioners. Its simplicity has known limitations. It is sensitive to feature scaling and to noisy small area rates. A single capacity parameter also restricts how well complex interactions can be represented. These challenges are familiar from small area labour market measurement, where rate estimates are noisy and unevenly precise across areas, and methods that borrow strength or model spatial dependence are often recommended (Benedetti et al. 2024). In this thesis, k-NN serves as a simple and interpretable engineering baseline rather than as a full small area estimator.

Random forest regression addresses several of these limitations. It represents non-linearities and interactions flexibly by aggregating many decorrelated trees, without requiring explicit functional assumptions. It is robust to monotone rescaling of inputs. Variable importance profiles provide practical guidance for feature engineering and policy interpretation, even though instance level explanations are less direct than in k-NN. In municipal prediction tasks with mixed feature scales and complex response surfaces, ensemble methods such as random forests typically provide stable and reliable baselines.

In this application, both models show similar out-of-sample performance and point to the same main patterns. The remaining unexplained variability is driven by the inherent complexity of youth unemployment and by the limited set of covariates available at municipal level, not by weaknesses in the algorithms

themselves. From an engineering perspective, this is a useful result. It shows that standard, well understood machine learning methods are sufficient to reveal the key conclusion in this data: upper secondary school density has only a modest association with municipal youth unemployment, and the current network appears broadly adequate under present conditions. Richer and more detailed features, for example spatial accessibility metrics or programme level attributes, would refine local predictions and help target marginal improvements, but they are unlikely to overturn the main finding that school distribution is not a dominant driver of municipal youth unemployment today.

6.3 Summary of key findings

This subsection answers the three research questions directly and then synthesises the main empirical insights.

The first research question asked how far better geographic and programme access is associated with lower municipal youth unemployment, after conditioning on key socio-economic covariates. The results indicate a small, directionally negative association between upper-secondary provider density and youth unemployment. Provider counts retain only limited explanatory power once scale and basic demographics are controlled for. In other words, denser provision is helpful but not decisive. At the current level of aggregation, the models suggest that the Finnish upper-secondary school network is broadly adequate with respect to municipal youth unemployment. There is no evidence of widespread structural undersupply, even though local vulnerabilities may exist.

The second question asked where and for which groups marginal improvements in accessibility are most strongly aligned with reduced youth unemployment, once non-linear responses and interaction effects are taken into account. Allowing for such structure improves fit modestly, which implies genuine complexity in how education supply, settlement patterns and labour demand interact. However, the estimated effects remain small and noisy. The analysis does not reveal large, robust pockets where marginal increases in institutional density would, by

themselves, generate substantial reductions in youth unemployment. Any such effects appear small and context dependent. Instead, the results point to conditional dependence mediated by commuting distances, socio-economic composition and programme–demand matching, especially in smaller municipalities where rate volatility is high.

The third question examined how planned vocational funding cuts might shift these relationships across regions if they are treated as a policy shock. Given the small marginal effect of provider density on youth unemployment, the model-based scenarios do not suggest large average changes in municipal unemployment rates. This remains true even under the planned cuts. At the same time, the scenarios highlight relative risks in municipalities with already sparse provision and long travel distances. In these areas, further reductions in programme capacity could erode the currently adequate level of access. In such areas, cuts may weaken school-to-work bridges and intensify the social and economic frictions described above, even if the aggregate national effect remains modest.

Beyond these direct answers, four additional findings are worth emphasising.

First, scale dominates: population size and population density carry much of the predictive signal. They reflect how settlement concentration co-moves with both service supply and labour-demand opportunities, and they dilute the stand-alone explanatory power of institutional density.

Second, non-linearities and interactions matter but do not overturn the main conclusion. They reveal that accessibility interacts with settlement structure and demand conditions, yet they leave a large share of variance unexplained.

Third, count-based indicators of provision capture availability but not programme mix, capacity, completion or travel-time frictions, all of which condition labour market relevance.

Fourth, data, rather than algorithms, is the main bottleneck. Without richer features, model-based guidance on fine-grained targeting of accessibility improvements will remain limited.

6.4 Practical implications and future research

The weak association between municipal provider counts and youth unemployment has several practical implications. First, it suggests that, at the municipal level, the current upper-secondary school network is roughly in balance with youth labour-market conditions. There is no sign that simply adding more institutions in most areas would, on its own, produce large reductions in youth unemployment. This conclusion is consistent with NEET research that finds public-service availability to be only weakly related to outcomes once individual factors and local demand are controlled (Ahlroth, Laaninen and Niemelä 2025).

Second, the analysis underscores the limitations of using institutional counts as proxies for access. Finnish municipalities cover large areas, and many students living near a boundary may have their nearest upper-secondary institution in a neighbouring municipality. Municipal counts therefore mismeasure true accessibility. A more robust assessment would require geocoded locations of institutions and residence-level coordinates for unemployed youth. The latter are not publicly available. Additional attenuation likely arises from unobserved labour-demand conditions, programme–labour-market mismatch, travel-time frictions and rate volatility in small denominators. Each of these factors dilutes relationships that are measured only with simple count-based indicators.

Third, the results confirm that model selection is not the main constraint. As Section 6.2 discussed, the k-nearest-neighbours and random forest regressors were tuned and validated using cross-validation and a held-out test split. Both handled non-linearity and interactions to the extent allowed by the available features. The modest out-of-sample fit is better explained by data limitations. These include small municipal samples, noisy rate outcomes in sparsely populated areas, and missing covariates such as socio-economic composition, programme mix and capacity, labour demand and travel-time accessibility. Spatial spillovers across municipal borders likely further weaken signal. The main bottleneck is therefore measurement and feature coverage rather than model choice.

These considerations support a new working null hypothesis (H0) for future research: conditional on travel-time accessibility, socio-economic composition and local labour demand, municipal institutional density has no independent association with youth unemployment. Testing this hypothesis would require a richer data infrastructure. A natural next step would be to estimate such models using spatial accessibility metrics, for example isochrone-based measures or two-step floating catchment area indices. Multilevel or spatial error/lag models, combined with cross-validated prediction, would also be appropriate. Implementing this would involve combining Opintopolku coordinates, road and transit networks, and secure microdata on residence locations where permissible.

From a policy perspective, the findings suggest that efforts to reduce youth unemployment should focus less on aggregate provider counts and more on three fronts. First, on aligning programme mix and capacity with local and regional demand, building on European evidence that well matched vocational provision improves early employment (CEDEFOP 2024; CEDEFOP 2025; E2 Research 2023). Second, on strengthening school-to-work bridges, including internships, apprenticeships and employer partnerships, especially in municipalities with sparse provision. Third, on mitigating the social and economic frictions of educational mobility, through targeted student support, mental health services and housing solutions that reduce the costs of leaving home. Under planned funding cuts, these priorities become more urgent in municipalities already close to the margin of adequate provision.

REFERENCES

Ahlroth, E., Laaninen, M. & Niemelä, M. 2025. Conditions of youth transition: individual and municipal factors related to the NEET rate in Finland, Journal of Youth Studies. Available at: https://doi.org/10.1080/13676261.2025.2520311. Search date 8.8.2025.

Athey, S. and Imbens, G. 2019. Machine learning methods that economists should know about. Annual Review of Economics, 11. Available at: https://www.annualreviews.org/content/journals/10.1146/annurev-economics-080217-053433. Referenced 13.8.2025.

Benedetti, R., Coli, M., Fabrizi, E., Marchetti, S. & Salvati, N. 2024. Handling Out-of-Sample Areas to Estimate the Unemployment Rate at Local Labour Market Area, International Statistical Review. Available at: https://doi.org/10.1111/insr.12596. Search date 16.9.2025.

Beckett, S. 1983. Worstward Ho. John Calder. London.

Breiman, L. 1996. Bagging predictors. Machine Learning, 24 (2). Springer, New York.

Breiman, L. 2001. Random forests. Machine Learning, 45 (1). Springer, New York.

Caruana, R. and Niculescu-Mizil, A. 2006. An empirical comparison of supervised learning algorithms. Proceedings of the 23rd International Conference on Machine Learning, pp. 161–168. Available at: https://dl.acm.org/doi/10.1145/1143844.1143865. Search date 9.4.2025.

CEDEFOP 2017. Briefing note: Looking back to look ahead: preparing the future of VET in Europe 2020–30. Available at: https://www.cedefop.europa.eu/en/publications/9123. Search date 6.11.2025.

CEDEFOP 2024. VET Data insights: what is happening with vocational education and training? Available at: https://www.cedefop.europa.eu/files/2024_28_02_post_2020_vet_policy_15_data_insghts_v11.pdf. Search date 6.11.2025.

CEDEFOP 2025. What is new in IVET, key pointers from statistics. https://www.cedefop.europa.eu/en/data-insights/what-new-ivet-key-pointers-statistics. Search date 8.11.2025.

Chen, X., & Ishwaran, H. 2012. Random forests for genomic data analysis. Genomics, 99 (6). Available at: https://doi.org/10.1016/j.ygeno.2012.04.003. Search date 4.7.2025.

Chernozhukov, V., Chetverikov, D., Demirer, M., Duflo, E., Hansen, C., Newey, W. and Robins, J. 2018. Double/debiased machine learning for treatment and structural parameters. Econometrics Journal, 21. Available at: https://academic.oup.com/ectj/article/21/1/C1/5056401. Search date 9.11.2025.

Cover, T. & Hart P. 1967. Nearest neighbor pattern classification. IEEE Transactions on Information Theory. Stanford IT-Forum IT-11. Palo Alto.

Dangeti, P. 2017. Statistics for Machine Learning: Techniques for exploring supervised, unsupervised, and reinforcement learning models with Python and R. Packt Publishing Ltd. Birmingham.

E2 Research 2023. New research on international talents in Finland, foreign degree students enjoy living in Finland but nearly half plan to leave after graduating. E2 Research. Available at: https://www.e2.fi/en/news/all-news/press-release-new-research-on-international-talents-in-finland-foreign-degree-students-enjoy-living-in-finland-but-nearly-half-are-likely-to-leave-finland-after-completing-their-studies.html. Search date 8.11.2025.

Friedman, J., Hastie, T. & Tibshirani, R. 2009. The Elements of Statistical Learning. Second Edition. Springer. Stanford.

Géron, A. 2017. Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly Media, Inc. Sebastopol.

Guerrero-Puerta, L. & Guerrero, M. 2023. Exploring the relationship between early leaving of education and training and mental health among youth in Spain. Societies, 13. Available at: https://www.mdpi.com/2075-4698/13/5/103. Search date 9.5.2025.

Ho, T. 1998. The random subspace method for constructing decision forests. IEEE Transactions on Pattern Analysis and Machine Intelligence, 20 (8). IEEE. New York.

Häkkinen, I. 2002. The effect of student aid reform on graduation times, a duration analysis. IZA Summer School paper. Available at: https://legacy.iza.org/en/papers/summerschool/5_hakkinen.pdf. Search date 6.11.2025.

Hietala, V. 2025. Increase in student loans and its implications for Finnish students. Master's thesis, Aalto University. Available at: https://aaltodoc.aalto.fi/server/api/core/bitstreams/e87524e5-5a19-4f76-a4c6-709fa8266301/content. Search date 6.11.2025.

Jauhiainen, S. 2011. Overeducation in the Finnish regional labour markets. Journal of Socio-Economics, 40. Available at: https://www.sciencedirect.com/science/article/pii/S1056819023014021. Search date 8.11.2025.

Kela 2025a. Financial aid for students in Finland. Kela. Available at: https://www.kela.fi/students. Search date 8.11.2025.

Kela 2025b. Kela asks approximately 21,000 students to explain lack of progress in their studies, some may lose their benefits. Press release, 3 October 2025. Available at: https://www.kela.fi/news/kela-asks-approximately-21-000-students-to-explain-lack-of-progress-in-their-studies-some-may-lose-their-benefits. Search date 8.11.2025.

Kultalahti, O. 2001. Internal migration and specialising labour markets in Finland in the 1990s. Available at: https://journal.fi/nypr/article/view/44961/11241. Search date 3.8.2025

Järvinen, T. & Vanttaja, M. 2005. Nuorten työttömyys ja koulutuksen merkitys. Yhteiskuntapolitiikka 70 (2). Helsinki.

Kalenius, A. 2014. Koulutus, työllisyys ja työttömyys, Opetus- ja kulttuuriministeriön julkaisuja 2014:13. Helsinki.

Kelleher, J., Mac Namee, B. & D'Arcy A. 2015. Fundamentals Of Machine Learning For Predictive Data Analytics. MIT Press. Cambridge. Massachusetts.

Kleinberg, J. & Tardos, E. 2005. Algorithm design. PEARSON/Addison Wesley. Cornell University. Ithaca.

Kylliäinen, O. 2024. The academic and employment effects of study grant income limit raises. Master's thesis, Tampere University. Trepo. https://trepo.tuni.fi/bitstream/handle/10024/160646/Kylli%C3%A4inenOlavi.pdf.

Louppe, G. 2014. PhD dissertation: Understanding Random Forests, from Theory to Practice. Université de Liège.

Lundberg, S. and Lee, S. 2017. A unified approach to interpreting model predictions. Advances in Neural Information Processing Systems 30. Available at: https://proceedings.neurips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf. Search date 8.11.2025.

McMullin, P. 2021. Geographical mobility and non completion of upper secondary education in Finland and Germany. Research on Finnish Society. Helda. Available at: https://helda.helsinki.fi/bitstreams/a4907cbc-9bcb-4aac-8115-ce59b10cedb1/download. Search date 7.11.2025

Mullainathan, S. and Spiess, J. 2017. Machine learning, an applied econometric approach. Journal of Economic Perspectives, 31. Available at: https://www.aeaweb.org/articles?id=10.1257/jep.31.2.87. Search date 3.10.2025.

Nietzsche, F. 1883. Also sprach Zarathustra: Ein Buch für Alle und Keinen. Band I, "Zarahustras Vorrede", § 5. Ernst Schmeitzner. Chmenitz.

Nilsson, N. 1998. Introduction to machine learning. Stanford University. Palo Alto.

OAJ 2024. Mitä ammatillisen koulutuksen 120 miljoonan euron leikkaus tarkoittaa käytännössä? OAJ Blogi. Available at: https://www.oaj.fi/ajankohtaista/blogiartikkelit/asiantuntijoiden-blogit/2024/mita-ammatillisen-koulutuksen-120-miljoonan-euron-leikkaus-tarkoittaa-kaytannossa/. Search date 8.2.2025.

Prasad, A., Iverson, L., & Liaw, A. 2006. Newer classification and regression tree techniques: Bagging and random forests for ecological prediction. Ecosystems, 9. Springer. New York.

Quinlan, J. 1986. Induction of decision trees. Machine Learning 1 (1). Springer, New York.

Raschka, S., Liu, Y. & Mirjalili, V. 2022. Machine Learning with PyTorch and Scikit-Learn. Packt Publishing Ltd. Birmingham.

Reuters 2025. Finland's unemployment rate stays among EU's highest in September. Available at: https://www.reuters.com/business/finlands-unemployment-rate-stays-among-eus-highest-september-2025-10-21/. Search date 2.11.2025.

Ribeiro, M., Singh, S. and Guestrin, C. 2016. "Why should I trust you?", explaining the predictions of any classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Available at: https://arxiv.org/abs/1602.04938. Search date 25.9.2025.

Serra, M., Freitas, P. and Balcão Reis, A. 2021. The impact of commuting on higher education students' academic performance. Working paper, Nova SBE. Available at: https://asset2021.sciencesconf.org/367953/The_Impact_of_commuting_on_Higher_Education_Students_Academic_Performance_25_08.pdf. Search date 5.7.2025.

Sharma, P. 2024. Understanding Distance Metrics Used in Machine Learning. Available at: https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning. Search date 3.2.2025.

Sipilä, N., Kestilä, L. & Martikainen, P. 2011a. Koulutuksen yhteys nuorten työttömyyteen: Mihin peruskoulututkinto riittää 2000-luvun alussa? Yhteiskuntapolitiikka 76 (2). Helsinki.

Sipilä, J., Kestilä, L. & Martikainen, P. 2011b. Social determinants of health in adolescence: Family background, education and attitudes. Sosiaalilääketie-teellinen Aikakauslehti 48 (2). Helsinki.

Şuhubi E. 2003. Functional Analysis. Springer. Dordrecht.

Sundqvist, A., Malmberg-Heimonen, I., Stain, H. and Strandh, M. 2024. Are educational transitions related to young people's loneliness and mental health, a systematic review. International Journal of Adolescence and Youth, 29.

Available at: https://www.tandfonline.com/doi/full/10.1080/02673843.2024.2373278. Search date 30.10.2025.

Tuononen, T. 2019. Employability of university graduates. Doctoral dissertation, University of Helsinki. Available at: https://helda.helsinki.fi/bitstreams/7cb618ce-1ad1-445b-b912-a1868924ad84/download. Search date 26.10.2025.

Työ- ja elinkeinoministeriö (TEM) 2009. Nuorten työllisyys: Haasteet ja mahdollisuudet. Helsinki.

University of Turku 2025. Ten questions about the impact of moving on children. Aurora online magazine, 17 September 2025. Available at: https://www.utu.fi/en/news/aurora-online-magazine/10-questions-about-the-impact-of-moving-on-children. Search date 16.10.2025.

Uusitalo, V., Hynninen, S., Kallio, M., Kovanen, L., Lehmus, M. and Vanhala, P. 2024. Geographical and occupational mismatch in Finland over the recent era of economic crises. Talouspolitiikan arviointineuvosto, Memorandum 1 2024. Available at: https://talouspolitiikanarviointineuvosto.fi/wp-content/uploads/2024/11/tpan-memorandum-1-2024_Geographical-and-Occupational-Mismatch-in-Finland-over-the-Recent-Era-of-Economic-Crises.pdf. Search date 16.9.2025.

Varian, H. 2014. Big data: New tricks for econometrics. Journal of Economic Perspectives, 28 (2). American Economic Association. Nashville.

Wager, S. and Athey, S. 2018. Estimation and inference of heterogeneous treatment effects using random forests. Journal of the American Statistical Association, 113(523), 1228–1242. Available at: https://www.tandfonline.com/doi/abs/10.1080/01621459.2017.1319839. Search date 22.10.2025.

Wikipedia Contributors 2019. Netflix Prize. Wikipedia. Available at: https://en.wikipedia.org/wiki/Netflix_prize. Search date 3.2.2025.

Witten, I., Frank, E., Hall, M. & Pal, C. 2016. Data Mining: Practical Machine Learning Tools and Techniques. Fourth edition. Kaufmann Publishers. Burlington.

APPENDICES

Walking through Random Forest algorithm with a toy dataset

Let's consider a small toy dataset with two features x_1 and x_2, and a target variable y:

Index	x_1	x_2	y
1	1	1	0
2	1	2	0
3	2	2	1
4	4	3	2
5	5	5	2
6	6	7	3
7	7	6	3

Phase 1: Bootstrap Sampling

A Random forest begins by drawing many bootstrap samples from the original dataset. Each sample is formed via random sampling with replacement, so individual observations can appear more than once within a sample, while some observations may be omitted entirely.

For simplicity, let's assume we create two bootstrap samples:

- **Bootstrap Sample 1:**
  - Index 1: (1, 1, 0)
  - Index 2: (1, 2, 0)
  - Index 3: (2, 2, 1)
  - Index 4: (4, 3, 2)
- **Bootstrap Sample 2:**
  - Index 2: (1, 2, 0)
  - Index 3: (2, 2, 1)
  - Index 4: (4, 3, 2)
  - Index 5: (5, 5, 2)

Phase 2: Building Decision Trees

For each bootstrap replicate, a decision tree is fitted. At every internal node, a randomly selected subset of predictors is evaluated for the split, which is chosen to reduce the within-node variance of the response.

Decision Tree 1 (from Bootstrap Sample 1)

1. **Root Node:**
   - Consider all features x_1 and x_2.
   - Find the best split to minimize the variance of y.
2. **First Split:**
   - Suppose the best split is on $x_1 \leq 1{,}5$.
   - Left Child: Index 1 and 2 ($x_1 = 1$)
   - Right Child: Index 3 and 4 ($x_1 = 2$ and $x_1 = 4$)
3. **Second Split (Left Child):**
   - Further split on $x_2 \leq 1{,}5$.
   - Left Leaf: Index 1 ($y = 0$)
   - Right Leaf: Index 2 ($y = 0$)
4. **Second Split (Right Child):**
   - Further split on $x_2 \leq 2{,}5$.
   - Left Leaf: Index 3 ($y = 1$)
   - Right Leaf: Index 4 ($y = 2$)

Decision Tree 2 (from Bootstrap Sample 2)

1. **Root Node:**
   - Consider all features x_1 and x_2.
   - Find the best split to minimize the variance of y.
2. **First Split:**
   - Suppose the best split is on $x_2 \leq 2{,}5$.
   - Left Child: Index 2 and 3 ($x_2 = 2$)
   - Right Child: Index 4 and 5 ($x_2 = 3$ and $x_2 = 5$)
3. **Second Split (Left Child):**
   - Further split on $x_1 \leq 1{,}5$.
   - Left Leaf: Index 2 ($y = 0$)
   - Right Leaf: Index 3 ($y = 1$)
4. **Second Split (Right Child):**
   - Further split on $x_1 \leq 4{,}5$.
   - Left Leaf: Index 4 ($y = 2$)
   - Right Leaf: Index 5 ($y = 2$)

Phase 3: Making Predictions

To make a prediction for a new data point, each tree in the forest makes a prediction, and the final prediction is the average of all the individual tree predictions.

New Data Point: $(3{,}3)$

1. **Decision Tree 1 Prediction:**

- Traverse the tree:
  - $x_1 = 3 > 1{,}5 \rightarrow$ Right Child
  - $x_2 = 3 > 2{,}5 \rightarrow$ Right Leaf
  - Predicted $y = 2$

2. **Decision Tree 2 Prediction:**
   - Traverse the tree:
     - $x_2 = 3 > 2{,}5 \rightarrow$ Right Child
     - $x_1 = 3 \leq 4{,}5 \rightarrow$ Left Leaf
     - Predicted $y = 2$
3. **Final Prediction:**
   - Average of predictions: $\frac{2+2}{2} = 2$

Figure A1-1 shows a plot of the datapoints. The color of the points indicates the y-value and the new datapoint is drawn as a cross. Figure A1-2 gives the Python code to create the plot.

Figure A1-3 shows the prediction sufrace of the 2-tree demonstration run and Figure A1-4 gives the Python code to create the plot.

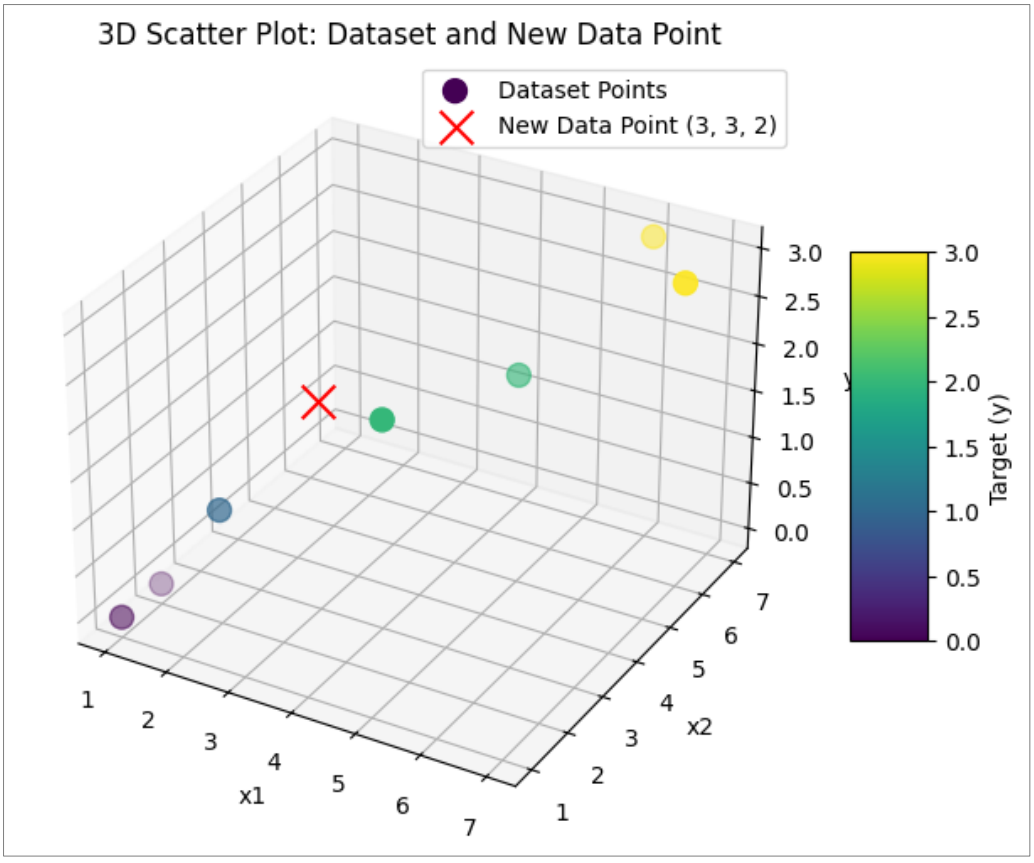

FIGURE A1-1. A scatter plot of the points in the dataset and the new data point

```
Code begins.

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Dataset
x1 = [1, 1, 2, 4, 5, 6, 7]  # Feature 1
x2 = [1, 2, 2, 3, 5, 7, 6]  # Feature 2
y  = [0, 0, 1, 2, 2, 3, 3]  # Target

# Create a 3D scatter plot
fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

# Plot the dataset points
scatter = ax.scatter(x1, x2, y, c=y, cmap='viridis', s=100, label='Dataset
                     Points')

# Add a new data point with predicted y-value [3,3,2]
new_x1, new_x2, new_y = 3, 3, 2
ax.scatter(new_x1, new_x2, new_y, c='red', marker='x', s=200, label='New Data
           Point (3, 3, 2)')

# Add labels and title
ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
ax.set_title('3D Scatter Plot: Dataset and New Data Point')

# Add a color bar
cbar = fig.colorbar(scatter, ax=ax, shrink=0.5, aspect=5)
cbar.set_label('Target (y)')

# Add a legend
ax.legend()

# Show the plot
plt.show()

Code ends.
```

FIGURE A1-2. Python code to create the plot in Figure A1-1.

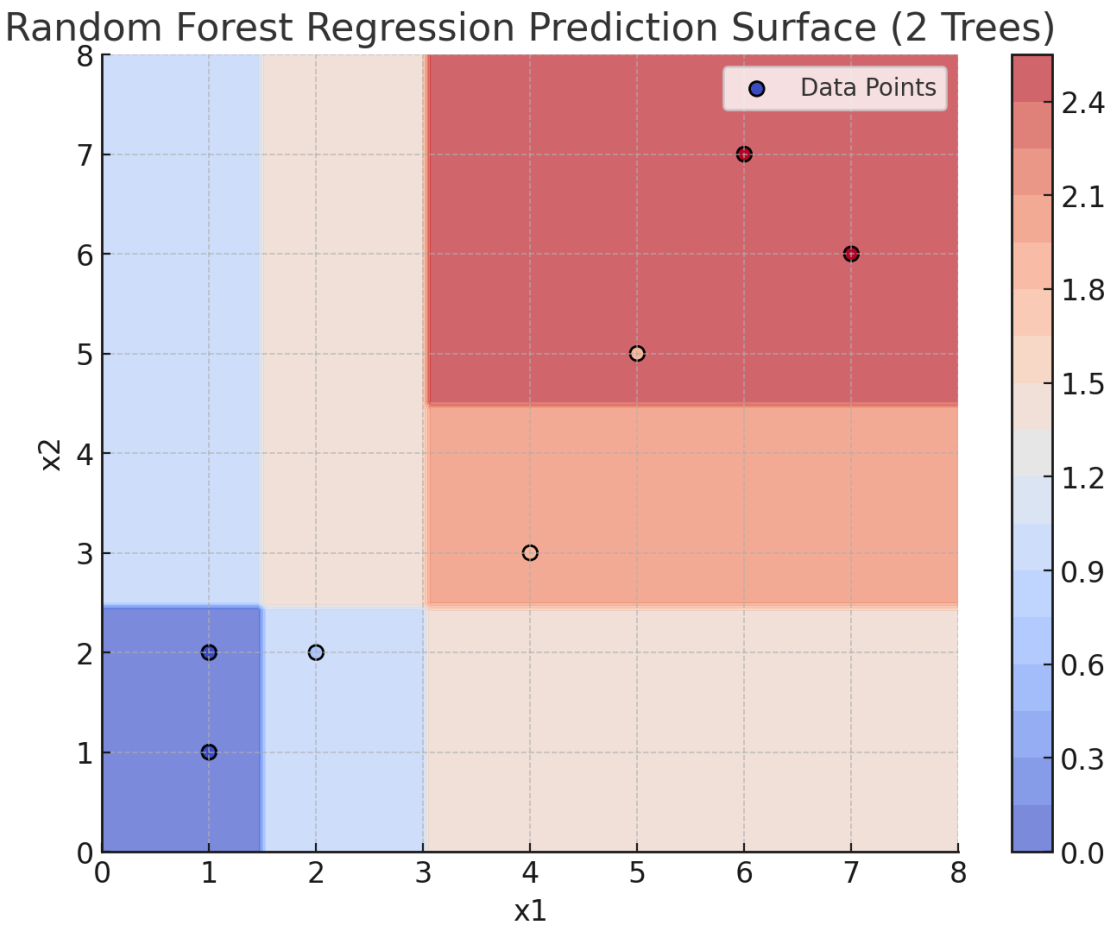

FIGURE A1-3. The prediction surface of the worked 2-tree example.

```
Code begins.

import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Define the dataset
x1 = np.array([1,1,2,4,5,6,7])
x2 = np.array([1,2,2,3,5,7,6])
y = np.array([0, 0, 1, 2, 2, 3, 3])

# Combine x1 and x2 into a feature matrix
X = np.column_stack((x1, x2))

# Train a Random Forest Regressor with 2 trees
rf = RandomForestRegressor(n_estimators=2, random_state=42)
rf.fit(X, y)

# Create a grid for visualization
x1_range = np.linspace(min(x1)-1, max(x1)+1, 100)
x2_range = np.linspace(min(x2)-1, max(x2)+1, 100)
x1_grid, x2_grid = np.meshgrid(x1_range, x2_range)
X_grid = np.column_stack((x1_grid.ravel(), x2_grid.ravel()))

# Predict the values for the entire grid
predictions = rf.predict(X_grid)
predictions = predictions.reshape(x1_grid.shape)

# Plot the prediction surface
plt.figure(figsize=(8, 6))
plt.contourf(x1_grid, x2_grid, predictions, levels=20, cmap='coolwarm', alpha=0.6)
plt.colorbar(label='Predicted y')

# Plot the training points
plt.scatter(x1, x2, c=y, edgecolors='black', cmap='coolwarm', s=100, label='Training
Data')
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Random Forest Regression (2 Trees) Prediction Surface')
plt.legend()
plt.show()

Code ends.
```

FIGURE A1-4. Python code to create the plot in Figure A1-3.

Walking through k-NN algorithm with a toy dataset

Let's consider a small dataset with two features x_1 and x_2, and a target variable y.

TABLE A2-1. Toy dataset.

Index	x_1	x_2	y
1	1	1	0
2	1	2	0
3	2	2	1
4	4	3	2
5	5	5	2
6	6	7	3
7	7	6	3

Choose the value of *k*

The first step in k-NN regression is to choose the number of neighbors k. For simplicity, let's choose $k = 3$.

1. Define a New Data Point

Suppose we have a new data point with features x_1=3 and x_2=3, and we want to predict the target value y for this point.

2. Calculate Distances

To find the nearest neighbors, we need to calculate the distance between the new data point and all the points in the dataset. A common distance metric is the Euclidean distance, which is calculated as:

$$Distance = \sqrt{\left(x_1^{new} - x_1^{old}\right)^2 + \left(x_2^{new} - x_2^{old}\right)^2}$$

Distance to point 1 (1, 1): $\sqrt{(3-1)^2+(3-1)^2} = \sqrt{8} \approx 2{,}828$

Other distances between data points 2-7 and the new point are calculated similarly.

3. Identify the k-Nearest Neighbors

The calculated distances are summarised in the Table A2-1. The table is sorted by distance in ascending order.

TABLE A2-2. Distances between new data point and dataset points.

Index	***Distance***	***y***
4	1,000	2
3	1,414	1
2	2,236	0
1	2,828	0
5	2,828	2
6	5,000	3
7	5,000	3

The three nearest neighbors are:

1. **Point 4:** Distance = 1,000, y = 2
2. **Point 3:** Distance = 1,414, y = 1
3. **Point 2:** Distance = 2,236, y = 0

4. Predict the Target Value

In k-NN regression, the predicted target value $\hat{y}$ for the new data point is the average of the target values of the k-nearest neighbors:

$$\hat{y} = \frac{y_1 + y_2 + y_3}{3} = \frac{0+1+2}{3} = 1$$

So, the predicted value for the new data point $(3{,}3)$ is $\hat{y} = 1$.

Figure A2-1 shows a plot of the datapoints. The color of the points indicates the y-value and the new datapoint is drawn as a cross. The 3 nearest neighbor points to the new data point identified by the algorithm are drawn as large blue circles. Figure A2-2 gives the Python code to create the plot.

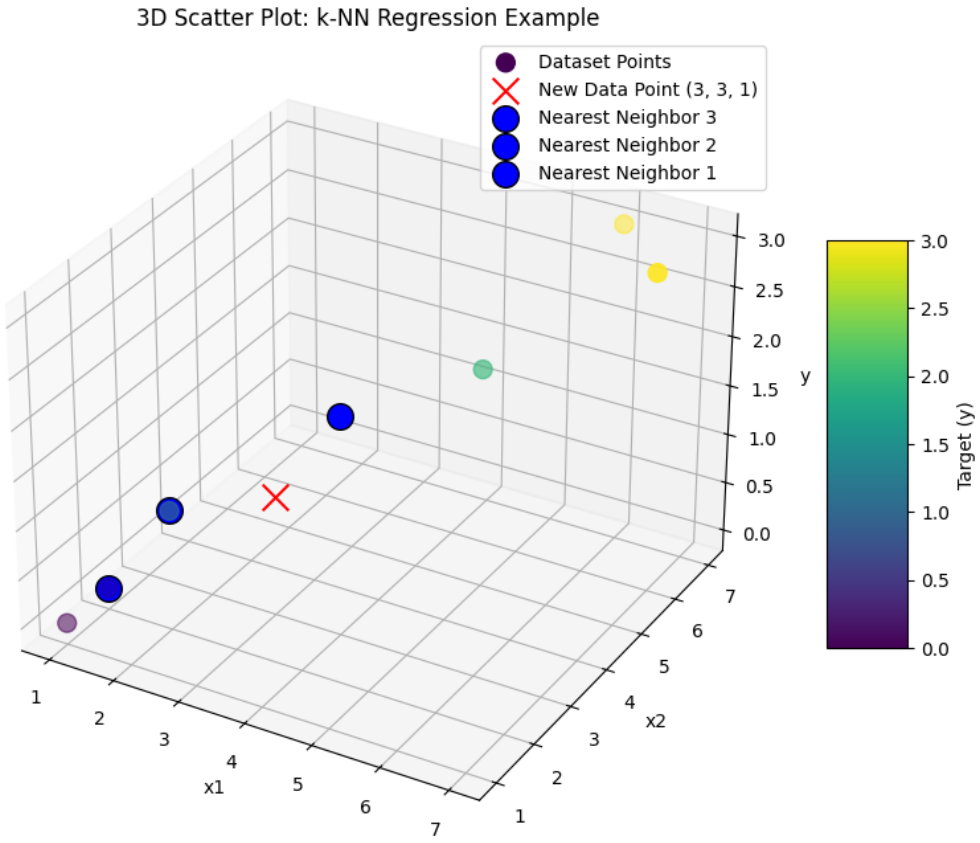

FIGURE A2-1: A scatter plot of the points in the dataset and the new data point

```
Code begins.

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np

# Dataset
x1 = [1, 1, 2, 4, 5, 6, 7]  # Feature 1
x2 = [1, 2, 2, 3, 5, 7, 6]  # Feature 2
y =  [0, 0, 1, 2, 2, 3, 3]  # Target

# New data point
new_x1, new_x2, new_y = 3, 3, 1  # Point (3,3) with predicted y=1

# k-NN: Identify the k-nearest neighbors (k=3) using Euclidean distances
distances = np.sqrt((np.array(x1) - new_x1)**2 + (np.array(x2) - new_x2)**2)
nearest_indices = np.argsort(distances)[:3] # Indices of the 3 nearest neighs

# Create a 3D scatter plot
fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')

# Plot the dataset points
scatter = ax.scatter(x1, x2, y, c=y, cmap='viridis', s=100, label='Dataset ')

# Highlight the new data point
ax.scatter(new_x1, new_x2, new_y, c='red', marker='x', s=200, label='New Data
           Point (3, 3, 1)')

# Highlight the k-nearest neighbors
for i in nearest_indices:
   ax.scatter(x1[i], x2[i], y[i], c='blue', marker='o', s=200,
              edgecolors='black', label=f"Nearest Neighbor {i}")

ax.set_xlabel('x1')
ax.set_ylabel('x2')
ax.set_zlabel('y')
ax.set_title('3D Scatter Plot: k-NN Regression Example')

# Add a color bar
cbar = fig.colorbar(scatter, ax=ax, shrink=0.5, aspect=5)
cbar.set_label('Target (y)')

ax.legend()

# Show the plot
plt.show()

Code ends.
```

FIGURE A2-2. Python code to create the plot in Figure A2-1.

The code to retrieve the list of upper-secondary institutions from Opintopolku's public Konfo service for every Finnish municipality is in Figure A3-1. The service returns the results as JSON; each item describes an institution, its provider, and any listed locations. The records were then cleaned, obvious duplicates were removed, and each institution was placed into one of three groups: general upper secondary (lukio), vocational, or other. A detailed file with one row per institution was produced, and a second file was created that sums the numbers by municipality and category. The outputs were saved as standard CSV files for easy reuse later in the analysis.

```
Code begins.

# -*- coding: utf-8 -*-
"""
Fetches institutions per municipality from Opintopolku's Konfo backend (public, no
auth), classifies them (General upper secondary / Vocational institutions / Other), and
builds a municipal summary.

Outputs:
- oppilaitokset_kunnittain_konfo.csv (detail, one row per institution)
- oppilaitokset_kunnittain_yhteenveto.csv (municipal summary: General, Vocational, Ot-
her, Total)

1) Download a CSV from the StatFin 125k table that contains the column "Alue" (PxWeb ->
Download -> CSV).
2) Set the STATFIN_125K_CSV path below.
3) Run: python retrieve_konfo_oppilaitokset.py
"""

import csv
import math
import time
import locale
import requests
import pandas as pd
from typing import List

# ---------- CONFIGURABLE ----------
STATFIN_125K_CSV = r"d:\OAMK\Thesis\syksy25\statfin_125k.csv"
KONFO_BASE = "https://opintopolku.fi/konfo-backend"
KONFO_SEARCH_PATH = "/search/oppilaitokset"
PAGE_SIZE = 200
# Aggregate areas that are not municipalities
NON_MUNICIPAL_LABELS = {
    "Koko maa", "Manner-Suomi", "Ahvenanmaa",
    "Suuralueet", "Maakunnat", "Seutukunnat", "ELY-keskukset",
    "Kunnat aakkosjärjestyksessä 2025", "Kunnat aakkosjärjestyksessä 2024",
    "Kunnat aakkosjärjestyksessä 2023", "Kunnat aakkosjärjestyksessä 2022",
}
def smart_read_statfin_csv(csv_path: str) -> pd.DataFrame:
    """
    Read a PxWeb CSV using multiple encodings and separators. If only one
```

```
    column appears and its header contains commas, retry with a comma as the
    separator.
    """
    encodings = ["utf-8-sig", "utf-8", "cp1252", "latin-1"]
    seps = [",", ";", "\t", "|"]
    last_err = None

    for enc in encodings:
        for sep in seps:
            try:
                df = pd.read_csv(csv_path, sep=sep, encoding=enc, dtype=str, en-
gine="python", quotechar='"')
                df.columns = [c.strip().lstrip("\ufeff") for c in df.columns]
                # wrong separator case
                if df.shape[1] == 1:
                    only_col = df.columns[0]
                    sample = df.iloc[0, 0] if not df.empty else ""
                    if ("," in only_col) or (isinstance(sample, str) and "," in
sample):
                        df2 = pd.read_csv(csv_path, sep=",", encoding=enc, dtype=str,
engine="python", quotechar='"')
                        df2.columns = [c.strip().lstrip("\ufeff") for c in df2.columns]
                        return df2
                return df
            except Exception as e:
                last_err = e
                continue
    raise RuntimeError(f"CSV read failed. Last error: {last_err}")

def finnish_sort(seq: List[str]) -> List[str]:
    """Sort according to Finnish collation (PyICU -> locale -> fallback)."""
    try:
        from icu import Collator, Locale  # pip install PyICU
        coll = Collator.createInstance(Locale("fi_FI"))
        return sorted(seq, key=coll.getSortKey)
    except Exception:
        pass
    try:
        locale.setlocale(locale.LC_COLLATE, "fi_FI.UTF-8")
        return sorted(seq, key=locale.strxfrm)
    except Exception:
        return sorted(seq)

def load_municipalities_from_statfin(csv_path: str) -> List[str]:
    """Extract municipalities from the StatFin CSV column 'Alue' and sort alphabeti-
cally."""
    df = smart_read_statfin_csv(csv_path)

    # Ensure 'Alue' exists (fallback to English 'Area')
    if "Alue" not in df.columns:
        candidates = [c for c in df.columns if c.lower().strip() in ("alue", "area")]
        if not candidates:
            raise RuntimeError("CSV does not contain column 'Alue' (or 'Area').")
        df = df.rename(columns={candidates[0]: "Alue"})

    areas = df["Alue"].dropna().astype(str).str.strip()
    areas = areas[~areas.isin(NON_MUNICIPAL_LABELS)]
    municipalities = sorted(set(a for a in areas if a))
```

```
    municipalities = finnish_sort(municipalities)
    return municipalities

def safe_get(url: str, params: dict, tries: int = 5) -> requests.Response:
    """GET with backoff for 429/503 conditions."""
    delay = 1.2
    for _ in range(tries):
        r = requests.get(url, params=params, timeout=30)
        if r.status_code == 200:
            return r
        if r.status_code in (429, 503):
            time.sleep(delay)
            delay *= 1.8
            continue
        raise RuntimeError(f"HTTP {r.status_code}: {r.text[:200]}")
    raise RuntimeError("Too many retries on 429/503")

def _collect_typestring_from_hit(hit: dict) -> str:
    """
    Build a 'type-text' string from many possible fields in a hit
    for classification (resilient to alternative keys).
    """
    parts = []

    # Individual fields or lists of type keys
    for key in ("oppilaitostyyppiKoodi", "oppilaitostyyppi", "oppilaitostyypit",
"type", "types", "tyypit"):
        val = hit.get(key)
        if not val:
            continue
        if isinstance(val, dict):
            # e.g. {"koodi":"11","nimi":{"fi":"Lukiot",...}}
            parts.append(str(val))
            name = val.get("nimi") or val.get("name")
            if isinstance(name, dict):
                parts.append(name.get("fi") or name.get("sv") or name.get("en") or "")
            else:
                parts.append(str(name))
            code = val.get("koodi") or val.get("code")
            if code:
                parts.append(str(code))
        elif isinstance(val, list):
            parts.extend([str(x) for x in val])
        else:
            parts.append(str(val))

    return " ".join([p for p in parts if p]).lower()

def classify_institution(hit: dict, name: str | None) -> str:
    """
    Return class: 'Lukiot' / 'Ammatilliset oppilaitokset' / 'Muut'
    - first try to infer from type (codes/names),
    - if that fails, fall back to name keywords.
    """
    tblob = _collect_typestring_from_hit(hit)
    name_lc = (name or "").lower()

    # Keywords for types (type text or name)
```

```
    if any(k in tblob for k in ["lukio", "gymnasium", "upper secondary general"]) or
"lukio" in name_lc:
        return "Lukiot"

    if any(k in tblob for k in ["ammatill", "vocational", "yrkes", "yrkesutbildning",
"ammattiopisto"]) or \
       any(k in name_lc for k in ["ammatill", "ammattiopisto", "yrkes", "vocational"]):
        return "Ammatilliset oppilaitokset"

    # (Additional special cases could be added here, but the user requested a broad
'other' bucket)
    return "Muut"

def fetch_institutions_for_municipality(muni_name: str) -> list:
    """
    Fetch institutions from Konfo that have provision in the given municipality.
    - uses free-text 'keyword' = municipality
    - pagination via size/from
    - de-duplicates by OID or name
    - if response contains locations, ensure a municipality match
    - classifies into General / Vocational / Other
    """
    results = []

    params = {"keyword": muni_name, "size": PAGE_SIZE, "from": 0}
    r = safe_get(KONFO_BASE + KONFO_SEARCH_PATH, params)
    data = r.json()
    hits = data.get("hits", [])
    total = data.get("total", len(hits))
    results.extend(hits)

    pages = math.ceil(total / PAGE_SIZE)
    for p in range(1, pages):
        params["from"] = p * PAGE_SIZE
        r = safe_get(KONFO_BASE + KONFO_SEARCH_PATH, params)
        data = r.json()
        results.extend(data.get("hits", []))

    out = {}
    for h in results:
        oid = h.get("oid") or h.get("oppilaitosOid") or h.get("id") or ""
        name_obj = h.get("nimi") or h.get("name")
        if isinstance(name_obj, dict):
            name = name_obj.get("fi") or name_obj.get("sv") or name_obj.get("en")
        else:
            name = name_obj

        provider_obj = h.get("organisaatio") or h.get("provider")
        if isinstance(provider_obj, dict):
            provider = provider_obj.get("nimi") or provider_obj.get("name")
            if isinstance(provider, dict):
                provider = provider.get("fi") or provider.get("sv") or provi-
der.get("en")
        else:
            provider = provider_obj

        locations = h.get("locations") or h.get("sijainnit") or []
        if locations and muni_name:
```

```
            blob = " ".join([str(x) for x in locations]).lower()
            if muni_name.lower() not in blob:
                # no clear reference to the given municipality -> skip
                continue

        # classification
        category = classify_institution(h, name)

        key = oid or (name or "").strip().lower()
        if not key:
            continue
        item = out.setdefault(key, {
            "municipality": muni_name,
            "oppilaitos_oid": oid,
            "oppilaitos_nimi": name,
            "jarjestaja": provider,
            "category": category
        })
        if locations:
            existing = item.get("locations_text", "")
            concat = "; ".join(sorted(set([existing] + [str(x) for x in locations if
x])))
            item["locations_text"] = concat

    return list(out.values())

def main():
    # 1) Municipalities from the StatFin CSV
    municipalities = load_municipalities_from_statfin(STATFIN_125K_CSV)
    print(f"Ladattiin {len(municipalities)} kuntaa StatFin-CSV:stä.")
    print("Esimerkkikunnat:", municipalities[:10])

    # 2) Query Konfo per municipality (detail level)
    all_rows = []
    for muni in municipalities:
        try:
            rows = fetch_institutions_for_municipality(muni)
            print(f"{muni}: {len(rows)} oppilaitosta (tarjontaa kunnassa)")
            all_rows.extend(rows)
        except Exception as e:
            print(f"[WARN] {muni}: {e}")

    if not all_rows:
        print("Ei tuloksia Konfosta.")
        return

    # 3) Save detail level as CSV
    detail_cols = ["municipality", "oppilaitos_oid", "oppilaitos_nimi", "jarjestaja",
"category", "locations_text"]
    for r in all_rows:
        for c in detail_cols:
            r.setdefault(c, "")
    detail_path = "oppilaitokset_kunnittain_konfo.csv"
    with open(detail_path, "w", newline="", encoding="utf-8") as f:
        w = csv.DictWriter(f, fieldnames=detail_cols)
        w.writeheader()
        w.writerows(all_rows)
    print(f"OK (detalji) -> {detail_path}")
```

```
    # 4) DataFrame -> municipal summary (General/Vocational/Other/Total)
    df = pd.DataFrame(all_rows)
    # compute per-municipality counts by category
    counts = (df.groupby(["municipality", "category"])["oppilaitos_oid"]
                .nunique().reset_index(name="count"))
    # pivot to columns
    wide = (counts.pivot_table(index="municipality",
                               columns="category",
                               values="count",
                               aggfunc="sum",
                               fill_value=0)
                  .reset_index())

    # ensure columns exist
    for col in ["Lukiot", "Ammatilliset oppilaitokset", "Muut"]:
        if col not in wide.columns:
            wide[col] = 0

    # Overall total
    wide["Kaikki oppilaitokset yhteensä"] = (wide["Lukiot"] +
                                    wide["Ammatilliset
                                          oppilaitokset"]+ wide["Muut"])

    # sort by municipality name
    wide = wide.sort_values("municipality").reset_index(drop=True)

    # 5) Print head(20) and save the summary as CSV
    print("\nKuntakooste (head 20):")
    print(wide.head(20).to_string(index=False))

    summary_path = "oppilaitokset_kunnittain_yhteenveto.csv"
    wide.to_csv(summary_path, index=False, encoding="utf-8")
    print(f"OK (kooste) -> {summary_path}")

if __name__ == "__main__":
    main()

Code ends.
```

FIGURE A3-1. Python code to retrieve school data from StatFin.

The script that preprocesses a TE-Office CSV that arrives in quite a convoluted layout is in Figure A4-1. First, it performs light cleaning: semicolons are replaced with commas and quotation marks are removed; rows containing the phrase “tieto on salassapito” are dropped.

The core step is year propagation. In the raw export, a header-like line (e.g., Vuosi 2023) appears once and is followed by multiple data lines that implicitly belong to that year. The script scans lines sequentially; whenever it encounters a line starting with Vuosi, it updates a current_year variable. For each subsequent non-empty data line, it prefixes that line with current_year, effectively “spreading” the year label across all observations until the next Vuosi line is found. This transforms grouped blocks into a flat, row-per-observation structure where every record explicitly carries its year, which is essential for reliable parsing and analysis.

Finally, the ungrouped file is read with pandas, skipping initial non-data lines, and columns are named ["Vuosi", "Kunta", "Yhteensä", "Miehet", "Naiset"], yielding a tidy table ready for downstream processing.

```
Code begins.

# The TE-Office file needs a bit pre-processing because
# the format is somewhat convoluted

import pandas as pd

input_file = "thas_vv0022412141650241173418811758 4s.csv"
output_file = "te_office_output.csv"

with open(input_file, "r", encoding="latin 1") as infile, open(output_file, "w", enco-
ding="latin 1") as outfile:
    for line in infile:
        modified_line = line.replace(";", ",").replace('"', "")
        if modified_line.count("tieto on salassapito") == 0:
            outfile.write(modified_line)

with open("te_office_output.csv", "r", encoding="latin 1") as file:
    lines = file.readlines()

ungrouped_data = []
current_year = 0

for line in lines:
    line = line.strip()
```

```
    if line.startswith("Vuosi"):
        current_year = int(line.split()[1])
    elif line:
        ungrouped_data.append(f'{current_year}{line}')

with open("te_office_ungrouped.csv", "w", encoding="latin 1") as file:
    file.write("\n".join(ungrouped_data))

une = pd.read_csv(
    "te_office_ungrouped.csv",
    sep=",",
    header=None,
    encoding='latin 1',
    skiprows=6
)

une.columns = ["Vuosi", "Kunta", "Yhteensä", "Miehet", "Naiset"]

Code ends.
```

FIGURE A4-1. Python code to preprocess TE-Office dataset.

Figure A5-1 gives the python code to do the final combination of all data sources into one file. This script enriches a local institutions summary with municipal population and area figures from Statistics Finland's PXWeb API by connecting to https://pxdata.stat.fi/PXWeb/api/v1/fi/Kuntien_avainluvut/Kuntien_avainluvut__2021/laaja_alueaikasarjat_2021.px.

First, the CSV oppilaitokset_kunnittain_yhteenveto.csv is loaded and its municipality names are normalized to a consistent form. PXWeb metadata are then fetched to identify the latest available year and the exact codes for the "Population on 31 Dec. Using that metadata, a POST payload is constructed that requests those indicators for all municipalities, respecting the table's required dimension order.

The JSON response is converted into a tidy DataFrame by expanding the dimension categories and aligning each observation with its labels. Next, the data are pivoted to a wide format so that population, land area, and optional total area become separate columns. These geographic fields are renamed to canonical names and joined back to the institutions table on the normalized municipality key. Finally, the merged dataset is written to oppilaitokset_kunnittain_enriched_geo.csv.

```
Code begins.

# -*- coding: utf-8 -*-
"""
Fetches municipal population and areas from Statistics Finland's PXWeb API
and joins them to the institutions summary.

Source table: Kuntien avainluvut -> laaja_alueaikasarjat_2021.px
API: https://pxdata.stat.fi/PXWeb/api/v1/fi/Kuntien_avainluvut/Kuntien_avainlu-
vut__2021/laaja_alueaikasarjat_2021.px
(requests the following in Tiedot/Information: 'Väkiluku, 31.12.' + 'Maapinta-ala, km2'
and, if available, also 'Kokonaispinta-ala, km2')

Input:  oppilaitokset_kunnittain_yhteenveto.csv (column 'municipality' + count columns)
Output: oppilaitokset_kunnittain_enriched_geo.csv (+ print head(20))
"""

import json
import requests
import pandas as pd
import unicodedata
from pathlib import Path

# --- Paths (change if needed) ---
INSTITUTIONS_CSV = "oppilaitokset_kunnittain_yhteenveto.csv"
OUT_CSV = "oppilaitokset_kunnittain_enriched_geo.csv"

# PXWeb API table (Municipal key figures)
```

```
PX_URL =
"https://pxdata.stat.fi/PXWeb/api/v1/fi/Kuntien_avainluvut/Kuntien_avainluvut__2
021/laaja_alueaikasarjat_2021.px"

def normalize_name(s: str) -> str:
    if s is None:
        return ""
    s = unicodedata.normalize("NFC", str(s)).strip()
    s = " ".join(s.split())
    return s

def fetch_px_metadata(url: str) -> dict:
    r = requests.get(url, timeout=60)
    r.raise_for_status()
    return r.json()

def choose_latest_year(meta: dict) -> str:
    # Find the Year dimension and pick the newest (last value)
    dims = meta.get("variables", [])
    for d in dims:
        code = d.get("code", "").lower()
        if "vuosi" in code or d.get("text","").lower()=="vuosi":
            years = d.get("values", [])
            return years[-1]
    raise RuntimeError("The Year dimension was not found in metadata.")

def find_info_codes(meta: dict) -> dict:
    """
    Returns a dictionary selecting:
      - 'pop'  : Väkiluku, 31.12. (Population on 31 Dec)
      - 'land' : Maapinta-ala, km2 (Land area)
      - 'total': Kokonaispinta-ala, km2 (Total area)  (if present)
    Uses the Tiedot/Information dimension based on labels.
    """
    out = {}
    for d in meta.get("variables", []):
        if d.get("code","").lower() in ("tiedot","information"):
            labels = d.get("valueTexts") or d.get("values") or []
            values = d.get("values") or []
            # Build a mapping value -> label
            labmap = {values[i]: labels[i] if i < len(labels) else values[i] for i in
range(len(values))}
            # Find identifiers based on label text
            pop_key = next((k for k,v in labmap.items() if "Väkiluku" in v and "31.12"
in v), None)
            land_key = next((k for k,v in labmap.items() if "Maapinta-ala" in v and
"km2" in v), None)
            tot_key = next((k for k,v in labmap.items() if "Kokonaispinta-ala" in v and
"km2" in v), None)
            if not pop_key:
                # Fallback: any label containing 'Väkiluku'
                pop_key = next((k for k,v in labmap.items() if "Väkiluku" in v), None)
            if not land_key:
                # Sometimes marked without the '2' (e.g. km²)
                land_key = next((k for k,v in labmap.items() if "Maapinta-ala" in v),
None)
            if pop_key:
                out["pop"] = pop_key
```

```
            if land_key:
                out["land"] = land_key
            if tot_key:
                out["total"] = tot_key
            return out
    raise RuntimeError("Tiedot/Information dimension not found or identifiers could not
be inferred.")

def find_area_dimension(meta: dict) -> str:
    # Find the code of the Area dimension (often 'Alue 2021' / 'Alue' / 'Area 2021')
    for d in meta.get("variables", []):
        code = d.get("code","")
        text = d.get("text","")
        if any(k in code.lower() for k in ["alue", "area"]) or any(k in text.lower()
for k in ["alue","area"]):
            return d["code"]
    raise RuntimeError("Alue/Area dimension was not found.")

def build_query_payload(meta: dict, year: str, info_keys: dict, area_code: str) ->
dict:
    """
    Builds the POST payload:
      - Year  = newest
      - Info  = selected identifiers
      - Area  = all (values = meta.variables[area_code].values)
    """
    # Extract all area values
    area_values = None
    for d in meta["variables"]:
        if d["code"] == area_code:
            area_values = d["values"]
            break
    if not area_values:
        raise RuntimeError("Area value list is missing from metadata.")

    # PXWeb: the order of dimensions in the query must follow the table's variable or-
der
    q = []
    for d in meta["variables"]:
        code = d["code"]
        if code == area_code:
            q.append({"code": code, "selection": {"filter": "item", "values": area_va-
lues}})
        elif code.lower() in ("tiedot","information"):
            wanted = [info_keys[k] for k in ["pop","land"] if k in info_keys]
            if "total" in info_keys:  # total is optional
                wanted.append(info_keys["total"])
            q.append({"code": code, "selection": {"filter": "item", "values": wanted}})
        elif "vuosi" in code.lower() or d.get("text","").lower()=="vuosi":
            q.append({"code": code, "selection": {"filter": "item", "values": [year]}})
        else:
            # Potential other dimensions -> select all
            q.append({"code": code, "selection": {"filter": "all", "values": ["*"]}})

    payload = {"query": q, "response": {"format": "JSON"}}
    return payload

def px_to_dataframe(resp_json: dict) -> pd.DataFrame:
```

```
    """
    Converts PXWeb JSON into a tidy DataFrame (one row per observation).
    """
    ids = resp_json["id"]
    sizes = resp_json["size"]
    dims = resp_json["dimension"]

    # Precompute labels per dimension
    labels = []
    keys = []
    for dim_id in ids:
        cat = dims[dim_id]["category"]
        idx_codes = cat["index"]
        lab_map = cat.get("label") or cat.get("label", {})
        # If label mapping is missing, fallback to index codes themselves
        labs = [lab_map.get(k, k) if isinstance(lab_map, dict) else k for k in idx_co-
des]
        labels.append(labs)
        keys.append(idx_codes)

    # Cartesian product of positions
    from itertools import product
    pos = list(product(*[range(n) for n in sizes]))
    rows = []
    data_iter = iter(resp_json["data"])
    for p in pos:
        obs = next(data_iter)
        row = {}
        for i, dim_id in enumerate(ids):
            row[dim_id] = labels[i][p[i]]
        row["value"] = None if obs["values"][0] in (None, "NaN", "..", ".") else
float(obs["values"][0])
        rows.append(row)

    return pd.DataFrame(rows)

def main():
    # 1) Load the local institutions summary
    inst = pd.read_csv(INSTITUTIONS_CSV, encoding="utf-8")
    # Normalize the municipality name column
    if "municipality" not in inst.columns:
        # Try to find an alternative column name
        cand = [c for c in inst.columns if c.lower().strip() in ("kunta","municipa-
lity","alue","area","kommun")]
        if not cand:
            raise RuntimeError("Institutions CSV has no 'municipality' (or equivalent)
column.")
        inst = inst.rename(columns={cand[0]: "municipality"})
    inst["municipality"] = inst["municipality"].map(normalize_name)

    # 2) Fetch PXWeb metadata and decide selections
    meta = fetch_px_metadata(PX_URL)
    latest_year = choose_latest_year(meta)
    info_keys = find_info_codes(meta)  # {'pop': '...', 'land': '...', 'total':
'...'(optional)}
    area_code = find_area_dimension(meta)

    # 3) Build the query and fetch data
```

```
    payload = build_query_payload(meta, latest_year, info_keys, area_code)
    r = requests.post(PX_URL, json=payload, timeout=120)
    r.raise_for_status()
    data = r.json()

    df = px_to_dataframe(data)

    # 4) Clean columns: identify the name of the Area dimension column and the Info co-
lumn
    # Find the column whose id matches area_code
    area_col = next(c for c in df.columns if c == area_code or c.lo-
wer().startswith("alue") or c.lower().startswith("area"))
    info_col = next(c for c in df.columns if c.lower() in ("tiedot","information"))

    # 5) Pivot to wide form: columns for population/land area/optional total area
    wide = df.pivot_table(index=area_col, columns=info_col, values="value", agg-
func="first").reset_index()
    # Name the columns clearly: find labels containing population/area
    def find_col_like(sub: str):
        for c in wide.columns:
            if isinstance(c, str) and sub in c:
                return c
        return None

    col_pop = find_col_like("Väkiluku")
    col_land = find_col_like("Maapinta-ala")
    col_total = find_col_like("Kokonaispinta-ala")

    # Create canonical fields
    out = wide[[area_col]].copy()
    out["population"] = wide[col_pop] if col_pop else None
    out["land_area_km2"] = wide[col_land] if col_land else None
    if col_total:
        out["total_area_km2"] = wide[col_total]

    # 6) Join to the institutions summary by municipality name
    out = out.rename(columns={area_col: "municipality"})
    out["municipality"] = out["municipality"].map(normalize_name)

    merged = inst.merge(out, on="municipality", how="left")

    # 7) Save and show a small preview
    Path(OUT_CSV).parent.mkdir(parents=True, exist_ok=True)
    merged.to_csv(OUT_CSV, index=False, encoding="utf-8")

    print(f"OK. Latest year from PXWeb: {latest_year}")
    print(f"Rows in merged dataset: {len(merged)}")
    print("\nHead(20):")
    print(merged.head(20).to_string(index=False))
    print(f"\nSaved: {OUT_CSV}")

if __name__ == "__main__":
    main()

Code ends.
```

FIGURE A5-1. Python code to do the final dataset compilation.

Collected data has to be preprocessed before feeding it to machine learning algorithms. Figure A6-1 shows the code that does the EDA and preprocessing. Institution counts were read and municipality names were normalized to a common key through unicode canonicalization and whitespace cleanup.

Population was parsed from a Statistics Finland export some symbol and whitespace cleanup, and the newest year per municipality was retained. Youth unemployment counts were ingested from the TE dataset; suppressed values were set to missing, the latest year per municipality was selected, and numeric parsing was applied.

The histogram in Figure A6-2 shows how Finnish municipalities are distributed by youth unemployment rate, expressed as the number of young unemployed per 10,000 residents. Each bar counts how many municipalities fall within a given rate interval, revealing the overall level, spread, skew, and any extreme values. Because the rate is normalized per 10,000 people, the plot allows fair comparison across municipalities of very different sizes.

The employment statistics reveal some unexpected patterns. Figure A6-3 indicates that Jyväskylä records the highest relative youth unemployment rate, even though it is a prominent education centre in Central Finland with eighteen upper-secondary institutions and two universities; this juxtaposition suggests that educational presence alone does not guarantee favourable youth employment outcomes. By contrast, Figure A6-4 reports the lowest youth unemployment rates and includes several counterintuitive cases. The zero for Brändö reflects an actual zero in the underlying data, whereas the neighbouring municipality of Eckerö also records a zero but the data is marked “Values have been suppressed to protect confidentiality,” indicating statistical disclosure control rather than a true zero. Several remote eastern and north-eastern municipalities also appear among the best performers in this figure, despite high overall unemployment in the wider region. The present analysis does not establish whether these patterns reflect youth out-migration or other compositional effects.

All sources were joined on the normalized municipality name; population density, institutions per ten thousand residents, and the target rate of youth unemployed per ten thousand were derived. Missing values were profiled; outliers were flagged using the interquartile range rule on key variables and written to a separate table. A violin plot of the ouliers is in Figure A6-5.

Cleaned data were saved to a comma separated file, and a modeling subset with selected features and the target was persisted for subsequent analysis.

```
Code begins.

# EDA for oppilaitokset_kunnittain_enriched_geo.csv
# - Loads the merged CSV (no rebuild)
```

```
# - Derives features if missing
# - Data quality: NaNs, IQR outliers, saves an outliers CSV

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from pathlib import Path
from sklearn.preprocessing import StandardScaler

MERGED_PATH   = "oppilaitokset_kunnittain_enriched_geo.csv"
OUT_MODEL     = "knn_model_dataset.csv"
OUT_OUTLIERS  = "knn_outliers.csv"
FIG_DIR       = Path("figs_eda")
FIG_DIR.mkdir(exist_ok=True, parents=True)

# ---------- Load merged dataset ----------
merged = pd.read_csv(MERGED_PATH, encoding="utf-8")

# ---------- Derive features (if not already present) ----------
if "pop_density_per_km2" not in merged.columns and {"population","land_area_km2"}.is-
subset(merged.columns):
    merged["pop_density_per_km2"] = merged["population"] / merged["land_area_km2"]

if "institutions_per_10k" not in merged.columns and {"Kaikki oppilaitokset yh-
teensä","population"}.issubset(merged.columns):
    merged["institutions_per_10k"] = merged["Kaikki oppilaitokset yhteensä"] / (mer-
ged["population"]/10000.0)

if "target_rate_per_10k" not in merged.columns and {"youth_unemployed","popula-
tion"}.issubset(merged.columns):
    merged["target_rate_per_10k"] = np.where(
        merged["population"] > 0,
        merged["youth_unemployed"] / (merged["population"]/10000.0),
        np.nan
    )

# ---------- Data quality: NaNs + IQR outliers ----------
numeric_cols = [
    "Lukiot","Ammatilliset oppilaitokset","Muut","Kaikki oppilaitokset yhteensä",
    "population","land_area_km2","total_area_km2","pop_density_per_km2",
    "institutions_per_10k","youth_unemployed","target_rate_per_10k"
]
present_numeric = [c for c in numeric_cols if c in merged.columns]
nan_summary = merged[present_numeric].isna().sum().rename("NaN_count").to_frame()

iqr_cols = [c for c in [
    "target_rate_per_10k","youth_unemployed","Kaikki oppilaitokset yhteensä",
    "institutions_per_10k","pop_density_per_km2","population","land_area_km2"
] if c in merged.columns]

outlier_rows = []
outlier_meta = {}  # store IQR stats per metric for ranking
for c in iqr_cols:
    series = merged[c].dropna().astype(float)
    if len(series) < 5:
        continue
    q1 = series.quantile(0.25)
    q3 = series.quantile(0.75)
```

```
    iqr = q3 - q1
    lo = q1 - 1.5*iqr
    hi = q3 + 1.5*iqr
    mask = (merged[c] < lo) | (merged[c] > hi)
    flagged = merged.loc[mask, ["municipality", c]].copy()
    flagged["metric"] = c
    flagged["lo"] = lo
    flagged["hi"] = hi
    outlier_rows.append(flagged)

    # For ranking: count of outliers and max normalized exceedance beyond whisker
    if not flagged.empty:
        vals = flagged[c].astype(float).values
        exceed = np.where(vals < lo, lo - vals, vals - hi)  # distance outside whisker
        max_exceed = float(np.nanmax(exceed)) if exceed.size else 0.0
        norm_exceed = max_exceed / (iqr if iqr != 0 else 1.0)
        outlier_meta[c] = {
            "count": int(flagged.shape[0]),
            "max_exceed": max_exceed,
            "norm_exceed": norm_exceed
        }
    else:
        outlier_meta[c] = {"count": 0, "max_exceed": 0.0, "norm_exceed": 0.0}

outliers = (pd.concat(outlier_rows, ignore_index=True)
            if outlier_rows else
            pd.DataFrame(columns=["municipality","metric","value","lo","hi"]))

# Attach the actual outlier value for convenience
if not outliers.empty and "value" not in outliers.columns:
    outliers["value"] = np.nan
    for idx, row in outliers.iterrows():
        try:
            outliers.at[idx,"value"] = merged.loc[
                merged["municipality"]==row["municipality"], row["metric"]
            ].values[0]
        except Exception:
            pass

outliers.to_csv(OUT_OUTLIERS, index=False, encoding="utf-8")

# ---------- Modeling-ready export ----------
model_df = merged.dropna(subset=["target_rate_per_10k"]).copy()
feature_cols = [c for c in [
    "Lukiot","Ammatilliset oppilaitokset","Muut","Kaikki oppilaitokset yhteensä",
    "population","pop_density_per_km2","institutions_per_10k"
] if c in model_df.columns]

export_df = model_df[["municipality"] + feature_cols + ["target_rate_per_10k"]].copy()
export_df.to_csv(OUT_MODEL, index=False, encoding="utf-8")

# ---------- Correlation heatmap on standardized data ----------
X_all = export_df[feature_cols].astype(float).values
y_all = export_df["target_rate_per_10k"].astype(float).values

scaler_for_eda = StandardScaler()
X_scaled_for_eda = scaler_for_eda.fit_transform(X_all)
y_scaled_for_eda = StandardScaler().fit_transform(y_all.reshape(-1, 1)).ravel()
```

```
eda_df = pd.DataFrame(X_scaled_for_eda, columns=feature_cols)
eda_df["target_rate_per_10k"] = y_scaled_for_eda

corr_cols = feature_cols + ["target_rate_per_10k"]
corr = eda_df[corr_cols].astype(float).corr()

fig, ax = plt.subplots()
im = ax.imshow(corr.values, aspect="auto", vmin=-1, vmax=1)
ax.set_xticks(range(len(corr_cols)))
ax.set_yticks(range(len(corr_cols)))
ax.set_xticklabels(corr_cols, rotation=90)
ax.set_yticklabels(corr_cols)
ax.set_title("Correlation heatmap (Pearson) - standardized features and target")
cbar = fig.colorbar(im, ax=ax)
cbar.set_label("Pearson r")
fig.tight_layout()
fig.savefig(FIG_DIR / "eda_correlation_heatmap.png", dpi=150)
plt.show()

# ---------- Histograms for each feature and the target ----------
for c in feature_cols + ["target_rate_per_10k"]:
    if c not in export_df.columns:
        continue
    fig, ax = plt.subplots()
    ax.hist(export_df[c].dropna().values, bins=30)
    ax.set_title(f"Histogram: {c}")
    ax.set_xlabel(c)
    ax.set_ylabel("Count")
    fig.tight_layout()
    fig.savefig(FIG_DIR / f"eda_hist_{c}.png", dpi=150)
    plt.show()

# ---------- Feature vs target scatter plots ----------
for c in feature_cols:
    fig, ax = plt.subplots()
    ax.scatter(export_df[c].astype(float).values, export_df["target_rate_per_10k"].as-
type(float).values)
    ax.set_xlabel(c)
    ax.set_ylabel("target_rate_per_10k")
    ax.set_title(f"{c} vs target_rate_per_10k")
    fig.tight_layout()
    fig.savefig(FIG_DIR / f"eda_scatter_{c}_vs_target.png", dpi=150)
    plt.show()

# ---------- Ranked bar chart of target extremes ----------
top_n = 20
tmp = export_df[["municipality","target_rate_per_10k"]].dropna().copy()
tmp_sorted = tmp.sort_values("target_rate_per_10k", ascending=False)

# Top 20 highest
fig, ax = plt.subplots(figsize=(8, 6))
ax.barh(tmp_sorted.head(top_n)["municipality"][::-1], tmp_sorted.head(top_n)["tar-
get_rate_per_10k"][::-1])
ax.set_title(f"Highest {top_n} youth unemployment rates per 10k")
ax.set_xlabel("target_rate_per_10k")
fig.tight_layout()
fig.savefig(FIG_DIR / f"eda_bar_top_{top_n}_target.png", dpi=150)
```

```
plt.show()

# Bottom 20 lowest
fig, ax = plt.subplots(figsize=(8, 6))
ax.barh(tmp_sorted.tail(top_n)["municipality"], tmp_sorted.tail(top_n)["tar-
get_rate_per_10k"])
ax.set_title(f"Lowest {top_n} youth unemployment rates per 10k")
ax.set_xlabel("target_rate_per_10k")
fig.tight_layout()
fig.savefig(FIG_DIR / f"eda_bar_bottom_{top_n}_target.png", dpi=150)
plt.show()

# ---------- NEW: 3×2 violin plot of features with the most or largest outliers -------
---
# Rank features by: outlier count (primary), then normalized max exceedance (secondary)
if outlier_meta:
    ranking = (
        pd.DataFrame.from_dict(outlier_meta, orient="index")
          .assign(metric=lambda d: d.index)
          .sort_values(by=["count", "norm_exceed"], ascending=False)
    )
    top_features = ranking.head(6)["metric"].tolist()
else:
    # Fallback: choose up to 6 numeric features if no outliers detected
    top_features = [c for c in iqr_cols][:6]

# Create a single figure with 3x2 subplots
fig, axes = plt.subplots(2, 3, figsize=(12, 8))
axes = axes.flatten()

for i, c in enumerate(top_features):
    ax = axes[i]
    data = merged[c].dropna().astype(float).values
    if data.size > 0:
        vp = ax.violinplot(dataset=[data], showmeans=True, showextrema=True, showmedi-
ans=True)
        ax.set_title(c)
        ax.set_xticks([])  # single distribution per subplot
        ax.set_ylabel(c)
    else:
        ax.text(0.5, 0.5, "No data", ha="center", va="center")
        ax.set_title(c)
        ax.set_xticks([])
        ax.set_yticks([])

# If fewer than 6 features available, hide empty subplots
for j in range(len(top_features), 6):
    axes[j].axis("off")

fig.suptitle("Distributions of features with most or largest outliers (violin plots)")
fig.tight_layout(rect=[0, 0, 1, 0.96])
fig.savefig(FIG_DIR / "eda_violin_top_outlier_features.png", dpi=150)
plt.show()

# ---------- Console summary ----------
print("\n=== DATA QUALITY ===")
print("Rows in merged:", len(merged))
print("\nNaNs per numeric column:")
```

```
print(nan_summary.to_string())
if not outliers.empty:
    print("\nOutliers saved ->", OUT_OUTLIERS)
else:
    print("\nNo outliers detected with the IQR rule.")
print("\nModeling dataset saved ->", OUT_MODEL)
print("Figures saved ->", str(FIG_DIR.resolve()))

Code ends.
```

FIGURE A6-1. Python code for EDA preprocessing.

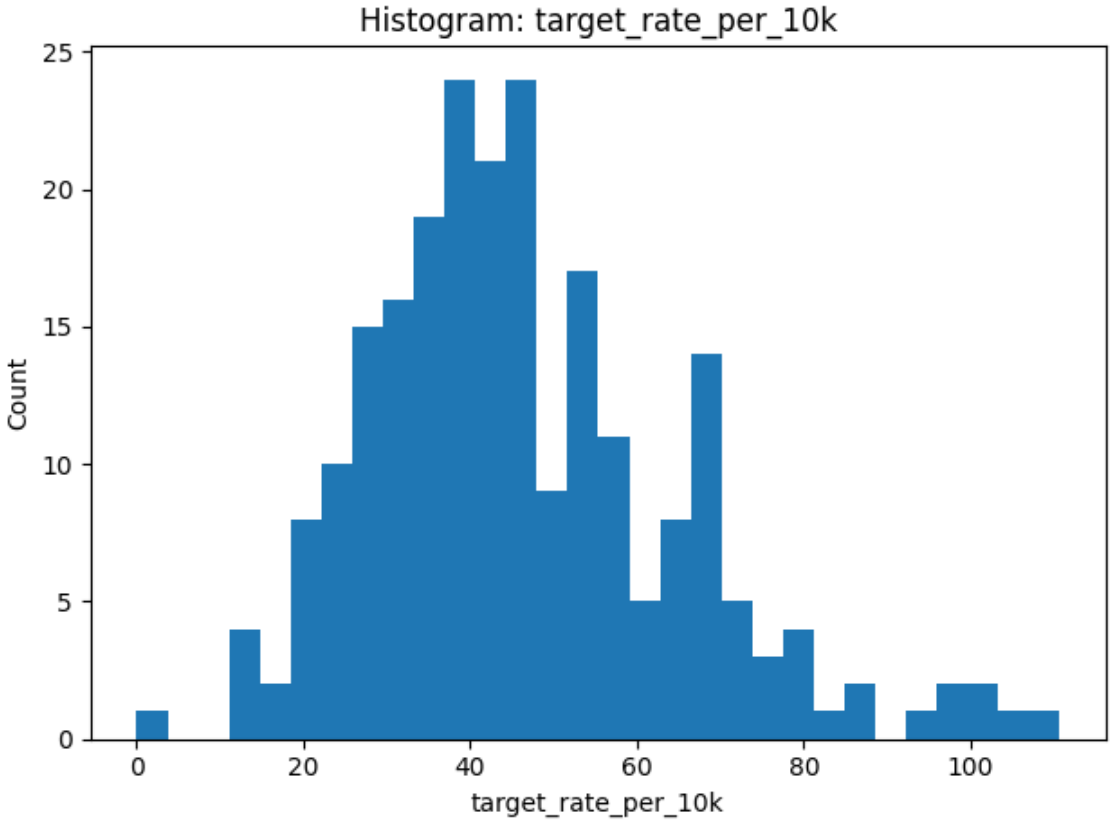

FIGURE A6-2. Histogram of the count of municipalities expressed as the number of young unemployed per 10,000 residents

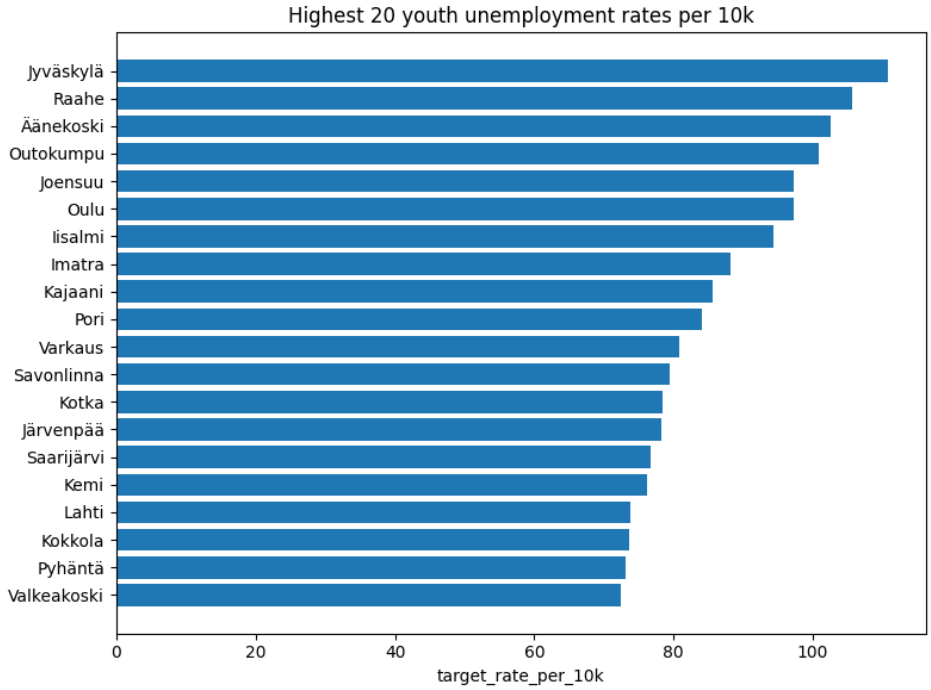

Figure A6-3. Highest youth unemployment rates of municipalities per 10,000 residents.

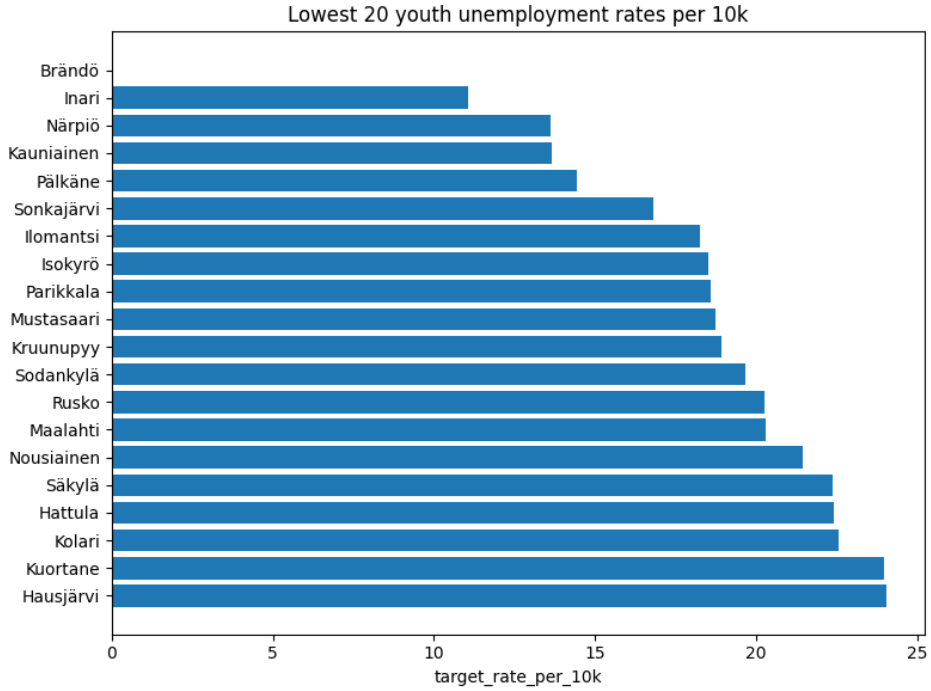

Figure A6-4. Lowest youth unemployment rates of municipalities per 10,000 residents.

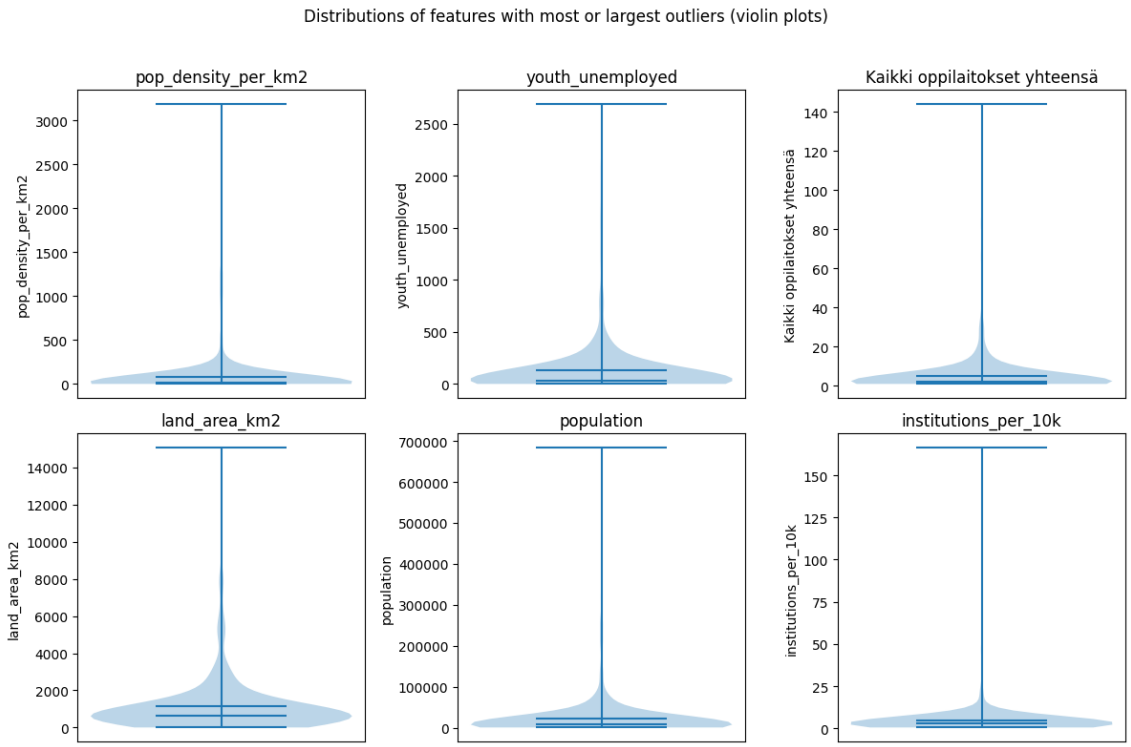

FIGURE A6-5. Violin plots of the features with most (or largest) outliers.

The k-NN code loads a prebuilt modeling table, splits it into training and test sets, and defines features and the target. It then searches for the best number of neighbors using 5-fold cross-validation in a leakage-safe pipeline that standardizes features before KNN. The results of the k-parameter after the crossvalidation are plotted in Figure A7-2. With the selected k, the code fits the final model on the training data, evaluates R^2 and RMSE on the test data, and produces two plots: mean CV R^2 versus k and predicted versus actual test values. Finally, it prints a brief results summary to the console.

Code begins.

```
# K-NN analysis only
# - Loads the modeling dataset saved by Cell 1
# - Runs 5-fold CV over k values with a leakage-safe pipeline (StandardScaler -> KNN)
# - Evaluates on a held-out test set
# - Plots: k vs CV R^2, predicted vs actual (test)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, KFold
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error

# Paths must match Cell 1
OUT_MODEL = "knn_model_dataset.csv"

# Load modeling dataset
df = pd.read_csv(OUT_MODEL, encoding="utf-8")

# Infer features/target from columns
all_cols = df.columns.tolist()
feature_cols = [c for c in all_cols if c not in ("municipality", "tar-
get_rate_per_10k")]
X_all = df[feature_cols].astype(float).values
y_all = df["target_rate_per_10k"].astype(float).values

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X_all, y_all, test_size=0.2, random_state=42
)

# Cross-validated k search (leakage-safe: scaler inside pipeline)
k_candidates = list(range(3, 26, 2))  # odd ks: 3..25
cv_scores = []
for k in k_candidates:
    pipe = Pipeline([
        ("scaler", StandardScaler()),
        ("knn", KNeighborsRegressor(n_neighbors=k, weights="distance"))
```

```
    ])
    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    scores = []
    for tr, va in kf.split(X_train):
        pipe.fit(X_train[tr], y_train[tr])
        pred = pipe.predict(X_train[va])
        scores.append(r2_score(y_train[va], pred))
    cv_scores.append(np.mean(scores))

best_idx = int(np.nanargmax(cv_scores)) if len(cv_scores) else 0
best_k = k_candidates[best_idx]
best_cv = cv_scores[best_idx] if cv_scores else np.nan

# Fit final model on full train and evaluate on test
final = Pipeline([
    ("scaler", StandardScaler()),
    ("knn", KNeighborsRegressor(n_neighbors=best_k, weights="distance"))
])
final.fit(X_train, y_train)
y_pred_test = final.predict(X_test)
r2 = r2_score(y_test, y_pred_test)
rmse = mean_squared_error(y_test, y_pred_test, squared=False)

# ---- Plots ----
plt.figure()
plt.plot(k_candidates, cv_scores, marker="o")
plt.xlabel("k (neighbors)")
plt.ylabel("Mean CV R^2 (5-fold)")
plt.title("KNN (Standardized): CV score vs k")
plt.tight_layout()
plt.show()

plt.figure()
plt.scatter(y_test, y_pred_test)
plt.xlabel("Actual (test) - youth unemployed per 10k")
plt.ylabel("Predicted (test) - youth unemployed per 10k")
plt.title(f"KNN (k={best_k}, standardized) - predicted vs actual (test)\nR^2={r2:.3f},
RMSE={rmse:.2f}")
lim = float(np.nanmax([y_test.max(), y_pred_test.max()]))
plt.plot([0, lim], [0, lim])
plt.tight_layout()
plt.show()

# ---- Console report ----
print("\n=== KNN RESULTS (Standardized pipeline) ===")
print("Features used:", feature_cols)
print(f"Best k: {best_k}, mean CV R^2: {best_cv:.3f}")
print(f"Test R^2: {r2:.3f}, RMSE: {rmse:.2f}")

Code ends.
```

FIGURE A7-1. Code to train and crossvalidate k-NN regressor.

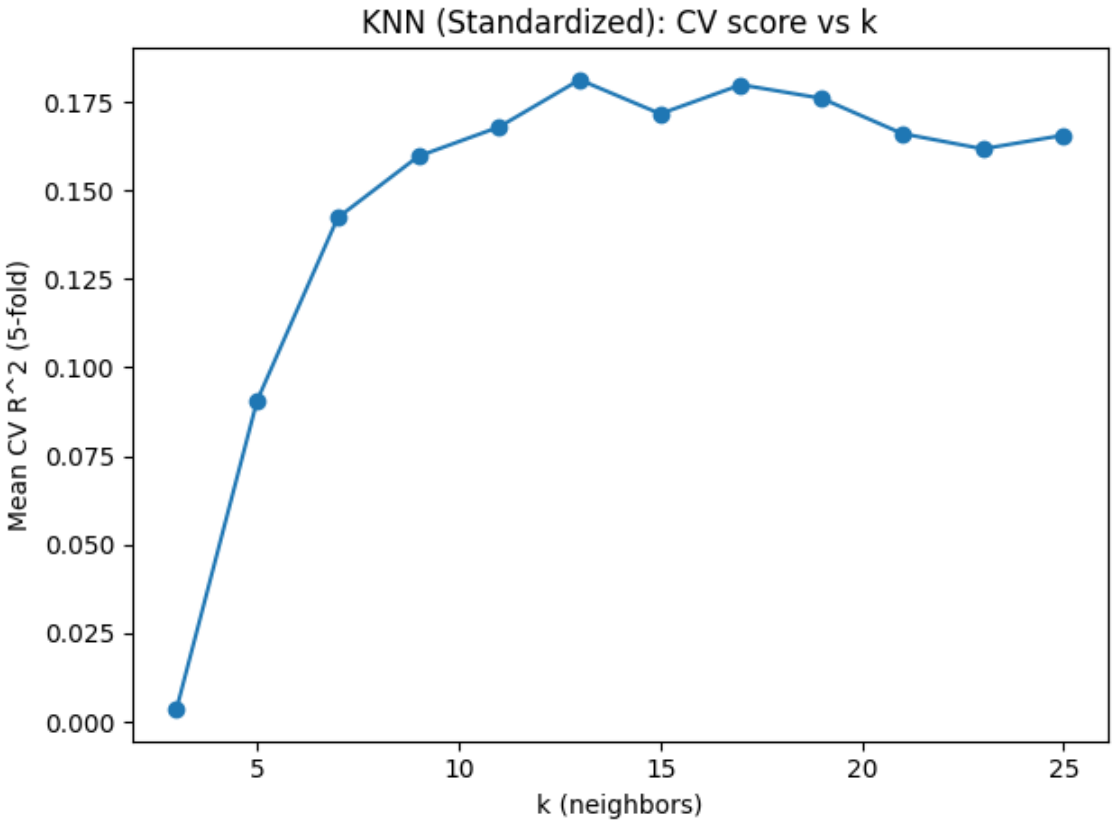

FIGURE A7-2. The R2 score with different k-values.

The script in Figure A8-1 applies a RandomForestRegressor to a pre-processed dataset, optimizing the model using a randomized search over multiple hyperparameters with 5-fold cross-validation. Despite the extensive hyperparameter tuning, the optimization process yielded no meaningful trend or consistent improvement, as the hyperparameter values exhibited considerable fluctuation without clear patterns. Figures A8-2 and A8-3 show outputs of two such parameter optimization runs. Following model training, the performance is evaluated using a held-out test set, and various diagnostic plots are generated, including validation curves, feature importances, residual plots, and learning curves. These visualizations provide insights into the model's accuracy and stability, offering a comprehensive view of its predictive capabilities.

```
Code begins.

import pandas as pd
import numpy as np
import re
import unicodedata
from pathlib import Path
from sklearn.model_selection import train_test_split, KFold, RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score, mean_squared_error
import matplotlib.pyplot as plt

# ---- Paths (edit if needed) ----
MERGED_PATH = "oppilaitokset_kunnittain_enriched_geo.csv"  # Preprocessed data from the
wrangling script
OUT_MODEL_DF  = "rf_model_dataset.csv"
OUT_PRED_TEST = "rf_test_predictions.csv"

def normalize_name(s: str) -> str:
    """Unicode-normalize and trim municipality names."""
    if s is None:
        return ""
    s = unicodedata.normalize("NFC", str(s)).strip()
    s = " ".join(s.split())
    return s

# ---------- Load pre-processed dataset (from the wrangling script) ----------
merged = pd.read_csv(MERGED_PATH, encoding="utf-8")

# ---------- Derived features (same as in the wrangling script) ----------
# Ensure features are present (already handled by the wrangling process)
if "pop_density_per_km2" not in merged.columns:
    merged["pop_density_per_km2"] = merged["population"] / merged["land_area_km2"]

if "institutions_per_10k" not in merged.columns:
    merged["institutions_per_10k"] = merged["Kaikki oppilaitokset yhteensä"] / (mer-
ged["population"]/10000.0)
```

```
if "target_rate_per_10k" not in merged.columns:
    merged["target_rate_per_10k"] = np.where(
        merged["population"] > 0,
        merged["youth_unemployed"] / (merged["population"]/10000.0),
        np.nan
    )

# ---------- Modeling dataset ----------
model_df = merged.dropna(subset=["target_rate_per_10k"]).copy()
feature_cols = [c for c in [
    "Lukiot", "Ammatilliset oppilaitokset", "Muut", "Kaikki oppilaitokset yhteensä",
    "population", "pop_density_per_km2", "institutions_per_10k"
] if c in model_df.columns]

X = model_df[feature_cols].astype(float).values
y = model_df["target_rate_per_10k"].astype(float).values

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran-
dom_state=42)

# ---------- Randomized hyperparameter search with 5-fold CV ----------
rf = RandomForestRegressor(random_state=42, n_jobs=-1)

param_dist = {
    "n_estimators": np.arange(100, 801, 50),     # 100..800
    "max_depth": [None] + list(np.arange(3, 31, 3)),
    "min_samples_split": [2, 5, 10, 20],
    "min_samples_leaf": [1, 2, 4, 8],
    "max_features": ["sqrt", "log2", None],      # None => all features
    "bootstrap": [True, False]
}

cv = KFold(n_splits=5, shuffle=True, random_state=42)

search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=80,              # increase if you want a wider search
    scoring="r2",
    n_jobs=-1,
    cv=cv,
    random_state=42,
    verbose=1
)

search.fit(X_train, y_train)

best_rf = search.best_estimator_
best_params = search.best_params_
best_cv_score = search.best_score_

# ---------- Evaluate on test set ----------
y_pred_test = best_rf.predict(X_test)
r2 = r2_score(y_test, y_pred_test)
rmse = mean_squared_error(y_test, y_pred_test, squared=False)

# Save modeling dataset and predictions
```

```
model_export = model_df[["municipality"] + feature_cols + ["tar-
get_rate_per_10k"]].copy()
model_export.to_csv(OUT_MODEL_DF, index=False, encoding="utf-8")

pred_df = pd.DataFrame({
    "municipality": model_df.iloc[y_test.argsort()].head(len(y_test))["municipa-
lity"].values,  # rough attach if needed
    "y_test": y_test,
    "y_pred_test": y_pred_test
})
pred_df.to_csv(OUT_PRED_TEST, index=False, encoding="utf-8")

# ---------- Plots ----------
# 1) Validation curves from CV results (mean test score vs key params)
results = pd.DataFrame(search.cv_results_)

# Helper to plot mean R^2 vs a single hyperparameter
def plot_cv_curve(param_name, sort_key=None):
    fig, ax = plt.subplots()
    dfp = results[[f"param_{param_name}", "mean_test_score"]].copy()
    dfp = dfp.dropna(subset=[f"param_{param_name}"])
    # Convert to numeric when possible
    try:
        dfp[f"param_{param_name}"] = pd.to_numeric(dfp[f"param_{param_name}"])
    except Exception:
        pass
    if sort_key is None:
        dfp = dfp.sort_values(by=f"param_{param_name}")
    else:
        dfp = dfp.sort_values(key=sort_key, by=f"param_{param_name}")
    ax.plot(dfp[f"param_{param_name}"].values, dfp["mean_test_score"].values, mar-
ker="o")
    ax.set_xlabel(param_name)
    ax.set_ylabel("Mean CV R^2")
    ax.set_title(f"CV curve: {param_name}")
    fig.tight_layout()
    plt.show()

# Plot a few key curves
plot_cv_curve("n_estimators")
plot_cv_curve("max_depth")
plot_cv_curve("min_samples_leaf")
plot_cv_curve("min_samples_split")

# 2) Feature importances (from the best RF)
fig, ax = plt.subplots()
importances = best_rf.feature_importances_
idx = np.argsort(importances)[::-1]
ax.bar(range(len(idx)), importances[idx])
ax.set_xticks(range(len(idx)))
ax.set_xticklabels([feature_cols[i] for i in idx], rotation=45, ha="right")
ax.set_ylabel("Importance")
ax.set_title("Random Forest feature importances")
fig.tight_layout()
plt.show()

# 3) Predicted vs actual (test)
fig, ax = plt.subplots()
```

```
ax.scatter(y_test, y_pred_test)
ax.set_xlabel("Actual (test) - youth unemployed per 10k")
ax.set_ylabel("Predicted (test) - youth unemployed per 10k")
ax.set_title(f"Random Forest - predicted vs actual (test)\nR^2={r2:.3f},
RMSE={rmse:.2f}")
lim = float(np.nanmax([y_test.max(), y_pred_test.max()]))
ax.plot([0, lim], [0, lim])
fig.tight_layout()
plt.show()

# 4) Residuals Plot (Predicted vs Residuals)
residuals = y_test - y_pred_test
fig, ax = plt.subplots()
ax.scatter(y_pred_test, residuals)
ax.axhline(0, color="red", linestyle="--")
ax.set_xlabel("Predicted values")
ax.set_ylabel("Residuals")
ax.set_title("Residuals plot: Predicted vs Residuals")
fig.tight_layout()
plt.show()

# 5) Learning Curves
train_errors, test_errors = [], []
train_sizes = np.linspace(0.1, 1.0, 10)
for train_size in train_sizes:
    X_train_sub, _, y_train_sub, _ = train_test_split(X_train, y_train,
train_size=train_size, random_state=42)
    best_rf.fit(X_train_sub, y_train_sub)
    train_errors.append(mean_squared_error(y_train_sub, best_rf.predict(X_train_sub),
squared=False))
    test_errors.append(mean_squared_error(y_test, best_rf.predict(X_test), squa-
red=False))

fig, ax = plt.subplots()
ax.plot(train_sizes, train_errors, label="Train Error")
ax.plot(train_sizes, test_errors, label="Test Error")
ax.set_xlabel("Training Set Size")
ax.set_ylabel("RMSE")
ax.set_title("Learning Curves (Random Forest)")
ax.legend()
fig.tight_layout()
plt.show()

# ---------- Console report ----------
print("\n=== RANDOM FOREST RESULTS ===")
print("Features used:", feature_cols)
print("Best params:", best_params)
print(f"Best CV R^2: {best_cv_score:.3f}")
print(f"Test R^2: {r2:.3f}, RMSE: {rmse:.2f}")
print(f"\nSaved files:\n- Modeling dataset: {OUT_MODEL_DF}\n- Test preds:
{OUT_PRED_TEST}")

Code ends.
```

FIGURE A8-1. Code to train and crossvalidate Random Forest regressor.

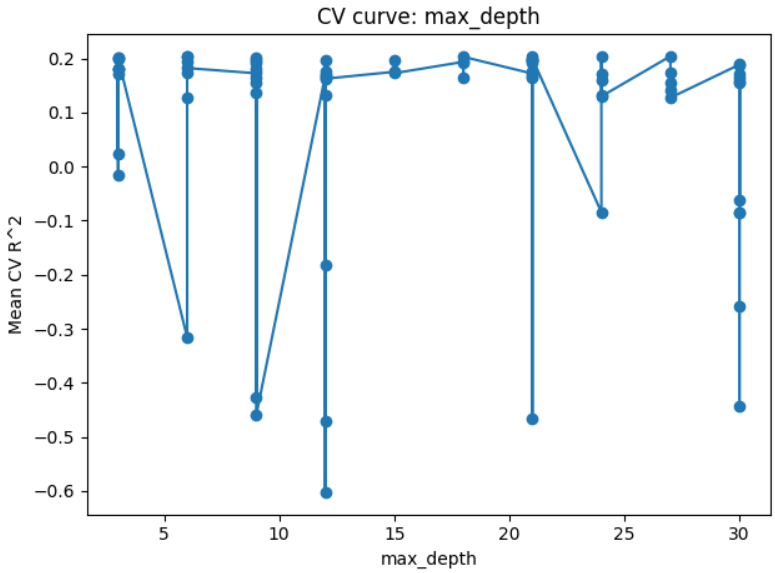

FIGURE A8-2. Performance plot of optimizing max_depth hyperparameter.

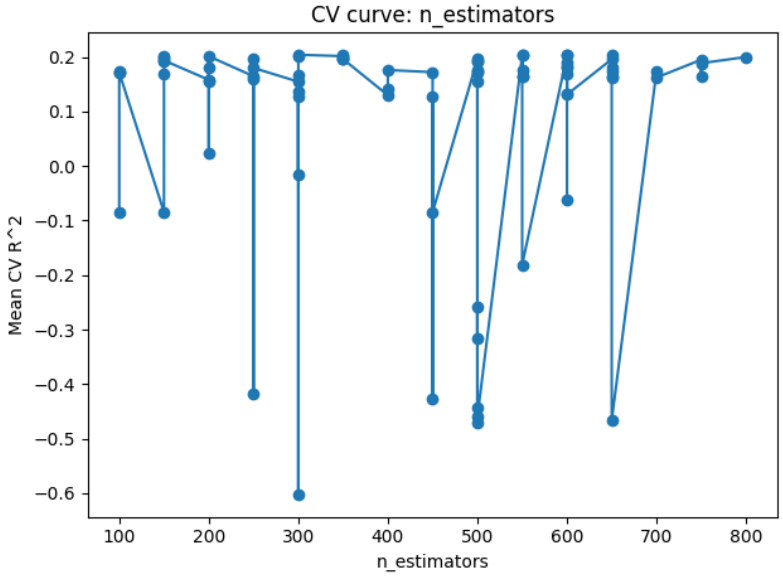

FIGURE A8-3. Performance plot of optimizing n_estimators hyperparameter.