A COMPHREHENSIVE REVIEW OF MACHINE LEARNING ALGORITHMS IN THE PUBLIC SECTOR

By Jamie Chan

SEPTEMBER 26, 2023
LOUGHBOROUGH UNIVERSITY

Contents

Abstract 4
Introduction (Chapter 1) 4
Literature Review (Chapter 2) 5
ML Algorithms in the Public Sector (2.1) 6
Selection Criteria (2.1.1) 6
Literature (2.1.2) 6
Critical Analysis (2.1.3) 9
Limitations of ML within the Public Sector (2.2) 10
Selection Criteria 2.2.1 11
Literature 2.2.2 11
Critical Analysis 2.2.3 13
Conclusion 2.3 14
Gaps in Literature 2.4 15
Methodology (Chapter 3) 15
Data Preparation and Cleaning 3.1 17
Merging 3.1.1 17
Missing Values 3.1.2 17
Outlier Removal 3.1.3 17
Normalising of Variables 3.1.4 17
Converting Categorical Values to Dummies 3.1.5 17
Constant Features 3.1.6 18
Feature Selection 3.2 18
Multicollinearity 3.3 18
Feature Selection Process 3.4 18
Gradient Boosted Decision Tree 3.4.1 19
Random Forest 3.4.2 19
Lasso Regularisation 3.4.3 19
Elastic Net Regularisation 3.4.4 19
K-Best 3.4.5 20
Variance Threshold 3.4.6 20
Individual ML algorithms 3.5 20
Linear Regression 3.5.1 20
Neural Network Regression 3.5.2 20
Decision Tree 3.5.3 20
Ensemble ML Algorithms 3.6 21

Bagging Regressor (Bootstrap Aggregating) 3.6.1 21
CatBoost Regressor 3.6.2 21
LightGBM (Light Gradient Boosted Machine) 3.6.3 21
Random Forest 3.6.4 21
Hyperparameter Tuning 3.7 22
Evaluation Metrics 3.8 22
Mean Absolute Error 3.8.1 22
R-Squared (R^2) 3.8.2 22
Results 3.9 23
Linear Regression 3.9.1 23
Decision Tree 3.9.2 23
Neural Network Regression 3.9.3 24
Bagging Regressor 3.9.4 24
Catboost Regressor 3.9.5 25
LightGBM 3.9.6 25
Random Forest 3.9.7 26
Computational Capacity Comparison 3.9.8 26
Analysis and Discussions (Chapter 4) 27
Feature Selection Analysis 4.1 27
Effects of Subsample Size on Algorithms 4.1.1 27
Related Research 4.1.2 27
Feature Interactions and Nonlinearity 4.1.3 28
Sparsity 4.1.4 28
Singular models 4.1.5 28
ML Algorithm Analysis 4.2 29
Related Research 4.2.1 29
High Dimensionality and Sparsity 4.2.2 30
Hyperparameter Analysis 4.2.3 30
Computational Requirements Ranking 4.3 31
Summary of Findings 4.4 32
Limitations 4.5 32
Dataset dependency 4.5.1 32
Hyperparameter Tuning and Computational Capabilities 4.5.2 33
Subsample Size and Computational Capabilities 4.5.3 33
Data Limitations 4.5.4 33
Adressing Computational Capacity Limitations 4.5.5 34

Conclusion (Chapter 5) 34
Theoretical Contributions 5.1 34
Alignment with Initial Expectation 5.2 35
Filling Research Gaps and Paving the Way for Future Research 5.3 35
Areas of Future Research 5.3.1 35
Closing Remarks 5.4 36
References 37
Appendix 42
Data Preparation and Cleaning (Figure 1.1 – 6.1) 42
Merging (Figure 1.1) 42
Missing Values (Figure 2.1 – 2.7) 42
Outlier Removal (Figure 3.1 – 3.6) 43
Normalising Variables (Figure 4.1) 45
Converting Categorical Values to Dummies (Figure 5.1) 45
Removing Constant Features (Figure 6.1) 45
Feature Selection (Figure 7.1 – 11.4) 45
Gradient Boosting (Figure 7.1 - 7.5) 45
Random Forest (Figure 8.1 – 8.5) 47
Lasso Regularisation (Figure 9.1 – 9.2) 48
Elastic Net (Figure 10.1 – 10.2) 49
K-Best (Figure 11.1 – 11.4) 50
Variance Threshold (Figure 12.1 – 12.2) 52
Individual ML Algorithms (Figure 13.1 – 15.3) 53
Linear Regression (Figure 13.1) 53
Neural Network Regression (Figure 14.1 – 14.2) 53
Decision Tree (Figure 15.1 – 15.3) 54
Ensemble ML Algorithms (Figure 16.1 – 19.2) 55
Bagging Regressor (Figure 16.1 – 16.2) 55
CatBoost Regressor (Figure 17.1 – 17.2) 56
LightGBM (Figure 18.1 – 18.2) 57
Random Forest (Figure 19.1 – 19.2) 58
Hyperparameter Tuning (Figure 20.1 – 20.6) 58
Neural Network Regression (Figure 20.1) 58
Decision Tree (Figure 20.2) 58
Bagging Regressor (Figure 20.3) 58
CatBoost Regressor (Figure 20.4) 59

LightGBM (Figure 20.5) 59
Random Forest (Figure 20.6) 59

Abstract

In today's data-driven world, the selection of the most suitable ML algorithm is critical for building accurate predictive models. This research presents a comprehensive comparative analysis of several ML algorithms, aiming to identify their strengths, weaknesses, and optimal use cases. The study focuses on comparing the performance of singular as well as ensemble approaches to investigate if more complex ensemble approaches are always superior to singular algorithms. The algorithms in question for this study are: Linear Regression (LR), Decision Trees (DT), Neural Network Regression (NN), CatBoost, LightGBM, Random Forest (RF) and Bagging Regressor (BR).

The research begins by formulating specific research objectives: to assess the predictive accuracy, computational efficiency, and scalability of these algorithms across a high-dimensional sparse dataset acquired from the official NHS database. The evaluation metrics encompass R^2 to measure the goodness of fit of the regression model and Mean Absolute Error (MAE) to measure the magnitude of errors of the prediction model. Methodologically, a diverse selection of feature selection methods is used on the full data to create subsamples of the original data. This is to reduce computational capacity and control for dataset dependency biases. The study applies each algorithm to the designated subsamples, fine-tuning hyperparameters and ensuring fair comparisons.

The findings reveal results that back the current literature: LightGBM was the most robust algorithm and consistently exhibited the best performance in R^2 and MAE across subsamples. This was closely followed by RF, then the other two ensemble algorithms, BR, CatBoost and finally the singular algorithms performed the worst. However computational efficiency and scalability analyses further highlight the trade-offs between these algorithms. Simpler algorithms experienced far less training time whilst ensemble approaches were longer. Despite this LightGBM had a relatively fast training time compared to its ensemble counterparts.

Keywords: Machine learning, algorithm, ensemble, singular, scalability, predictive modelling, interpretability, LightGBM, Random Forest, CatBoost, Bagging Regressor, Neural Network Regression, Decision Tree, Linear Regression.

Introduction (Chapter 1)

The public sector is posed with complex challenges, from energy consumption and healthcare to law enforcement and fraud detection, the demand for informed, data-driven decisions has never been greater. The expanding use of Machine Learning (ML) within the public sector has introduced and developed numerous unexplored opportunities for organisations in a variety of sectors. Traditional methods of predictive modelling are becoming near obsolete with the emergence of ML and with the growth of big data in the public sector, ML promises to reshape the way of predictive modelling Yeung & Lodge. (2019).

Across the globe, ML algorithms are sifting through terabytes of data, discerning hidden patterns, and illuminating paths to optimised resource allocation and improved services. Yet, as organisations embrace this technological revolution, numerous challenges arise.

- Which ML algorithms are most suitable for a given task or problem?
- How to perform feature selection to select the most relevant and informative features?
- Are ensemble algorithms always better than a singular algorithm?

These are the questions that organisations venturing into ML for predictive modelling are asking. In this paper the research will attempt to address these questions and build upon the existing literature within this domain. Thus, allowing organisations to better understand and utilise ML algorithms within their operations to improve their services and efficiency.

Additionally, gaps within the existing literature have been identified with a lack of literature regarding the limitations of scalability and computational resources associated with ML algorithms. Particularly research regarding the utilisation of complex ensemble algorithms with numerous layers and parameters within resource constrained settings. Additionally, there are gaps in the research associated with efficient learning algorithms which describe the development of new learning algorithms that require fewer iterations or samples to converge. Thus, reducing computational requirements of optimisation procedures. This research will include identifying and ranking high computational resource ML algorithms and consider the computational limitations of algorithms in addition to performance to provide a more comprehensive evaluation balancing the trade-off between performance and scalability.

There are already a vast host of studies assessing the comparison of ML algorithms and ensemble algorithms against singular algorithms. The existing literature often includes case studies and practical applications that compare ensembles with singular algorithms in real-world scenarios. In healthcare, ensembles are employed for disease diagnosis, patient risk assessment, and medical image analysis. Comparing the accuracy of ML algorithms can prove beneficial in early disease detection and improved patient care. In fraud detection, researchers and practitioners aim to develop models that can effectively identify fraudulent activities. In many cases fraud detection datasets comprise of highly imbalanced datasets which poses another challenge for ML within this industry. In energy consumption, the comparison of ensemble and singular ML algorithms enable organisations to make accurate predictions about energy usage, optimise energy management, reduce costs, and increase energy efficiency. However, organisations must be cognisant of potential challenges and limitations associated with ML. The very nature of the public sector regarding its diverse array of responsibilities, its need for transparency, and the critical consequences of its decisions, places unique demands on the application of ML Veale et al. (2018).

This research paper aims to investigate the uses and limitations of ML in the public sector. It delves into the success stories that have already begun to redefine the public sector and explores the potential for AI to reshape the future. Whilst also navigating the intricacies of privacy, ethics, and fairness that are integral to responsible ML implementation. Furthermore, this paper will decipher the complexities of ML model selection, hyperparameter tuning, and interpretability, offering guidance to practitioners striving for impactful and accountable solutions.

The content of this paper is not confined to theory; it extends into the real-world challenges faced by governments and public agencies. Including the analysis of case studies spanning healthcare predictions, fraud detection, and environmental concerns for energy consumption, revealing the tangible benefits of ML in diverse public contexts. By drawing insights from these examples, we illuminate the path forward for organisations seeking to harness the power of data-driven decision-making.

Literature Review (Chapter 2)

The motivation behind this literature review stems from the recognition that the selection of ML algorithms holds significant implications for the efficacy of public sector applications. The landscape of ML algorithms is vast, comprising a diverse array of techniques, ranging from classical models like

LR to advanced deep learning architectures. Each algorithm brings its unique strengths and limitations, making it imperative to assess their suitability for specific public sector use cases.

This literature review seeks to provide a comprehensive overview of existing research that compares and evaluates ML algorithms within the public sector. Additionally, it will explore studies that delve into the performance, scalability, interpretability, and ethical considerations associated with ML algorithm deployment in public sector domains. By synthesising the findings of previous research, this review aims to offer valuable insights into the factors that influence algorithm selection and the implications for decision-makers seeking to harness the potential of ML in the public sector. The structure of the literature review is split into two sections. The first section focuses upon the performance aspects of ML algorithms and its implementations in the public sector comparing various algorithms and how they are suited within different public sector domains. The second section provides an evaluation of the limitations associated with the implementation of ML in the public sector domain.

ML Algorithms in the Public Sector (2.1)

Within this theme of the literature review the purpose is to present research to address the following questions:

- How have ML algorithms been adopted within the public sector for predictive modelling purposes?
- What approaches have been taken regarding the comparison of performance in ML algorithms within the public sector?

Predictive modelling within the public sector holds significant importance as it offers the potential to improve decision-making, resource allocation, and service delivery, leading to more effective and efficient governance. The field of predictive modelling has undergone a profound revolution with the advent of ML. Traditional predictive modelling techniques, while valuable, often struggled to capture the intricate patterns and complex relationships inherent in modern datasets. ML has brought about a paradigm shift, equipping predictive modelling with powerful tools and algorithms that enable more accurate, flexible, and insightful predictions Ballester. (2021). This section of the literature review will explore the adoption of ML methods and approaches in the public sector for predictive modelling purposes.

Selection Criteria (2.1.1)

The selection criteria for the relevant research and literature were carried out by first identifying the key areas within the public sector that have adopted predictive modelling as a tool to improve their operations. After research, it was discovered that there was an abundance of research relating to ML for predictive modelling conducted in the industries of fraud detection, energy consumption and hospital readmissions. These areas were combined with the keywords ‘machine learning’, ‘predictive modelling’ and ‘algorithms’ also with various specific names of ML algorithms (e.g., ‘Logistic Regression’, ‘Random Forest’, ‘Neural Network’ etc) in Google scholar to identify relevant research. The research was chosen based on relevancy to answering the research question and other criteria such as the implementation of contemporary ML algorithms and the robustness of the experimental design.

Literature (2.1.2)

Fraud detection

In today’s rapidly evolving digital landscape, the prevalence of fraudulent activities has posed a substantial challenge for businesses and financial institutions. Traditional rule-based fraud detection systems are often insufficient to detect sophisticated and constantly evolving fraud patterns. As a

result, the integration of advanced ML algorithms has become crucial to predicting and preventing fraud effectively Sharma & Panigrahi. (2013).

In Sahin et al. (2011) the authors used classification models based upon Artificial Neural Networks (ANN) and Logistic Regression (LR) to design classification models to identify fraudulent cases associated with credit card fraud. Based upon their results the authors have concluded that ANN outperform LR models, citing that the LR models tended to overfit the training data more.

Khare et al. (2018) performed a comparison of ML methods which were trained on a dataset sourced from ULB Machine Learning Group which contained data on credit card transactions. They compared the performance of LR, Support Vector Machines (SVM), Decision Trees (DT) and Random Forests (RF). The authors concluded that the RF was the most precise and accurate (98.6% prediction accuracy). However, the authors preface that the RF algorithm tends to perform better on a larger volume of training data, and they also noted that the SVM algorithm struggled with training due to the low proportion of target fraudulent cases (0.173%).

In Patil et al. (2018) conducted a comparison of ML methods trained upon German credit card fraud data. They selected LR, DTs and RFs as their ML algorithms. The authors concluded that RF was the best performer in terms of accuracy, precision and recall parameters. They also noted that the RF may encounter issues relating to overfitting as data increases. In addition, the authors emphasise the importance of the nature of the data citing that data with a greater proportion of outliers will be trained more accurately on a DT as compared to LR as it's not as sensitive to outliers.

Tae & Hung. (2019) investigated the performance of ML algorithms on credit card fraud data. Firstly, they used data balancing approaches (over-sampling, under-sampling, both-sampling, ROSE and SMOTE) concluded that SMOTE produced the most desirable AUC. They then used a combination of ensemble and singular ML algorithms to train on the dataset. The authors compared the performance of (KNN, LR, NB, DT, RF, AdaBoost and NN). They concluded that RF performed the best based upon accuracy and precision. Also, the authors included hyperparameter tuning and training speeds within their evaluation.

Energy Consumption for Buildings (Environmental)

In the face of growing environmental concerns and the need for sustainable energy practices, optimising energy consumption in buildings has become an imperative. The advent of advanced ML algorithms has ushered in a new era of predictive modelling for energy consumption in buildings.

In Pham et al. (2020) the authors propose a RF based prediction model to predict the short-term energy consumption in the hourly resolution of multiple buildings. When compared to similar tree models such as Random Tree (RT) and M5P models the RF significantly outperformed its competing models in all performance measures (i.e., MAE, MAPE, RMSE, SI).

Guo et al. (2018) compared the performance of several ML methods regarding building a predictive model for energy demand for buildings. Within this study the authors develop and compare ML models including Multiple Linear Regression (MLR), Support Vector Regression (SVR), Extreme Learning Machine (ELM) and Backpropagation Neural Networks (BPNN). Due to the dataset being too vast the authors opted to utilise LASSO regularisation and correlation analysis to create different feature selections to train the various models on, this would ensure that the feature selection method would be controlled when comparing models. The authors concluded that ELM approach produced the best performance when trained with the feature selection method of selecting high correlation variables. In general, they concluded that the ELM and MLR models produced superior performance to that of the BPNN and SVR models. They concluded that the training performance was better on the BPNN and SVR models, but this did not carry over to unseen data suggesting

overfitting. Finally, the authors observed that the feature selection using LASSO regularisation performed better in general.

In Zeyu et al. (2018) they explore the benefits of using a homogenous ensemble approach to predict the hourly electricity usage of two educational buildings in North Central Florida. The chosen homogenous ensemble approach was RF as RF leverages the strengths of many DTs to create a more robust and accurate prediction. The authors compared the RF model with a Regression Tree (RT) and Support Vector Regression (SVR). The RF model had superior performances in all evaluation indices. The authors cite that the long span of data used within the training process included abnormal observations which inhibited the stability and accuracy of the RT.

Salam & Hibaoui. (2018) compared the ML algorithms, feedforward neural network, RF, DT and SVR. These ML algorithms were trained on a dataset detailing the power distribution network of a city located in Morocco. The authors applied a grid-search method to control for hyperparameter tuning and the authors concluded that RF had the least prediction errors in comparison to its counterparts.

Hospital Readmission

In the realm of healthcare, reducing hospital readmissions is a critical goal that not only improves patient outcomes but also optimises resource allocation and reduces healthcare costs. The integration of ML algorithms into predictive modelling for hospital readmissions has brought about transformative changes in patient care and management Huang et al. (2021).

Lo et al. (2021) developed and compared ML models in relation to predictive modelling of unplanned hospital readmissions within a 14-day time span. Four ML algorithms were used for comparison these were LR, RF, Extreme Gradient Boosting (XGBoost) and Gradient boosting with categorical features support (Catboost). Amongst the 4 ML models LR had the worst performance and Catboost had the best. It also must be noted that when training the LR model the features were reduced using VIF as multicollinearity was detected. An additional benefit of the Catboost model was that it was explainable and identified those features that were significant in relation to readmission rate.

Jiang et al. (2018) proposes a new framework for predictive modelling of hospital readmissions emphasising the importance of combining feature selection algorithms in combination with ML classification models. For their feature selection method the authors used an enhanced version of multi-objective bare-bones particle swarm optimisation (EMOBPSO) as their principal search strategy and a Greedy Local Search (GLS) was developed and merged into EMOBPSO to create their desired feature subset size. The ML models compared were SVM, RF and Deep Neural Network (DNN). They concluded that the DNN provided the most robust performance across various performance analysis metrics.

Min et al. (2019) investigated the cause of readmission within patients with Chronic Obstructive Pulmonary Disease (COPD) they trained and compared models on a real-world database containing medical data of 111,992 patients. The authors mainly focus upon the comparison of traditional (non-deep) ML methods to deep learning models. For the traditional method they chose to use LR, LR (with L1 and L2 regularisation), RF, SVM, Gradient Boosted Decision Tree (GBDT) and Multi-Layer Perceptron (MLP). For the deep learning ML models, they used Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and their variants (e.g., Long-Short Term Memory (LSTM) and Gated Recurrent Unit (GRU)). They concluded that the GBDT was the best performer amongst the traditional ML methods and that the deep learning methods did not outperform GBDT. Finally, the authors stress that knowledge driven feature selection in combination with data driven feature selection provided the most robust subsets for models to be trained on.

Saab et al. (2020) compared the efficacy of ML algorithms in relation to predicting hospital readmissions within a 30-day time period. They chose a range of supervised ML approaches for the comparison. The ML algorithms chosen were RF, DT, Boosting, ARR and LR. Amongst these algorithms ARR was superior with a mean prediction accuracy of (0.88) followed by LR (0.62), RF (0.62), DT (0.6), Boosting (0.55).

Public Sector Domain	Study	ML Algorithms Compared	Evaluation Metrics	Key Findings
Fraud Detection	Sahin et al. (2011)	ANN / LR	Accuracy / Precision	ANN outpefored LR, as LR tended to overfit
	Khare et al. (2018)	LR / SVM / DT / RF	Accuracy / Precision	RF was superior, worked well on high volume imbalanced data
	Patil et al. (2018)	LR / DT / RF	Accuracy / Precision	RF peformed best as it was most robust to outliers
	Tae & Hung. (2019)	KNN / NB / LR / DT / RF / AdaBoost / NN	Accuracy / Precision	RF peformed best
Energy Consumption	Pham et al. (2020)	RF / RT / M5P	MAE / MAPE / RMSE /SI	RF algorithm was superior
	Guo et al. (2018)	MLR / SVR / ELM / BPNN	MSE / MAE / R2	ELM and MLR were the most robust ML algorithms transfering well to unseen data
	Zeyu et al. (2018)	RF / RT / SVR	RMSE / R2 / MAE	RF peformed best
	Salam & Hibaoui. (2018)	FNN / RF / DT /SVR	Accuracy / Precision	RF has the least prediction errors
Hospital Readmission	Lo et al. (2021)	LR / RF / XGBoost / CatBoost	Recall / Precision / F1 Score / AUROC / AUPRC	CatBoost peformed best, it also showed good interpretability as ML algorithm
	Jiang et al. (2018)	SVM / RF /DNN	AUC / Accuracy / Precision / Recall	DNN was most robust algorithm
	Min et al. (2019)	RF / SVM / GBDT / MLP / CNN /RNN	Recall / Precision / F1 Score	GBDT was best, interestingly deep learning algorithms didn't outpeform GBDT
	Saab et al. (2020)	RF / DT / Boosting / ARR / LR	Accuracy / Precision / AUC	ARR was best and Boosting was the worst

Table of ML algorithm comparison studies

Critical Analysis (2.1.3)

Dataset / Feature Selection Dependency

The nature of a dataset has a huge influence on the performance of a specific ML algorithm Jesus et al. (2022). For example, a NN model with multiple hidden layers is more predisposed to capture non-linear patterns and relationships within a dataset as compared to a standard LR model. On the contrary, a dataset containing lots of linear relationships will be more suited to an LR model as a NN may tend to overfit. This represents the issue of dataset bias on the efficiency of ML algorithms within these types of studies. Hence, it is difficult to claim that one ML algorithm is universally better but is often the case that an algorithm suits the nature of the dataset as described within the 'No Free Lunch Theorem' by Wolpert & Macready. (1997).

Additionally, feature selection may heavily influence the performance of specific algorithms. For instance, different algorithms have varying sensitivities to the number of features. Feature selection allows you to find the right balance of model complexity for each algorithm, avoiding underfitting or overfitting. (Sahin et al. (2011), Pham et al. (2020), Lo et al. (2021)) showed little emphasis in their research to controlling feature selection methods. In order to mitigate this, researchers should choose a diverse range of subsamples using multiple different feature selection algorithms. Cross evaluating the results across a diverse range of subsamples will offer a broader perspective in evaluating algorithm performance.

Hyperparameter Tuning

Some ML algorithms are more reliant upon hyperparameter tuning to produce effective results. In general, ensemble methods are more reliant on proficient hyperparameter tuning. This is because

the combining of multiple models is accompanied by the complexity of ensuring that each base model contributes effectively and that their weaknesses are mitigated Radzi et al. (2021).

The choice of hyperparameters can be subjective and influenced by the researcher's bias. This can inadvertently favour certain algorithms that align with the researcher's expectations or preconceived notions, potentially leading to biased comparisons. In addition, hyperparameters are often interconnected, and tuning one might affect others. Finding the optimal combination of hyperparameters involves navigating complex trade-offs, and the interactions between them can be challenging to comprehend and optimise.

These limitations can be mitigated by adopting standardised procedures during the hyperparameter tuning process and ensuring that each algorithm receives a similar amount of tuning effort Cruz et al. (2021). Algorithms that have more hyperparameters might appear superior simply because they have more opportunities for tuning. A sensitivity analysis can also prove beneficial by varying hyperparameters slightly around the optimal values to gauge how robust the selected hyperparameters are to small changes.

Complexity and Interpretability

Majority of the studies focus solely upon performance of the models (Min et al. (2019), Jiang et al. (2018), Khare et al. (2018)). Also, in general more complex ensemble methods would outperform simpler approaches as the more intricate and sophisticated algorithms could more efficiently capture the patterns and relationships within the data Nikou et al. (2019). It should be noted that prediction efficiency is not the only evaluation metric, in some cases researchers may favour a less complex more interpretable algorithm. This may be because there are certain regulations surrounding the industry, they work in ensuring that the predictions from the model may be explainable and interpretable. In addition to this there may be computational limitations for ML practitioners who do not have the resources to run these large-scale sophisticated ML algorithms. Hence choosing a simpler algorithm which is comparable in performance to a more complex one may be the more viable decision.

Researchers could detail the entire methodology relevant to pre-processing and training the models so there is some insight on the processing complexity of using the algorithms. Perhaps include a ranking system to grade each algorithm on its run-time and computational resource requirements.

Limitations of ML within the Public Sector (2.2)

Within this section of the literature review the purpose is to present research to address the following questions:

- What are the main limitations of ML for predictive modelling within the public sector?
- How can limitations of ML be addressed?

As the volume and complexity of data continue to expand, the scalability of ML algorithms becomes a critical concern. The challenges related to computational resources, training time, data quality, and interpretability can hinder the scalability of ML algorithms. Addressing these limitations requires a multidisciplinary approach, involving advancements in hardware, algorithms, and data management techniques. Balancing scalability with the need for efficient models is a central challenge in the field of ML. The limited interpretability of complex models remains a significant limitation. Interpreting complex black-box models is crucial to ensure the responsible and ethical deployment of ML systems, especially in high-stakes applications where transparency, fairness, and accountability are paramount. Researchers and practitioners continue to work on developing techniques and tools for improving model interpretability while maintaining competitive performance, recognising that achieving a balance between the two is a central challenge in the field of ML.

Selection Criteria 2.2.1

The selection criteria for the scalability section considered a diverse range of scalability research, including, the comparison of ML algorithms based on their scalability potential, altering ML algorithms to enhance scalability and comparison of ML algorithms compatibility with parallel ML systems. Research was chosen based upon unique perspectives and insights as well as robust experimental designs and contribution to the field. Keywords such as, scalability, machine learning, big data and individual ML algorithm names were combined in Google Scholar to search for papers.

Regarding the interpretability section, it was identified through research that there was a substantial amount of existing literature focusing on the use of model-agnostic approaches within the healthcare domain as the healthcare domain is notoriously known for having stringent interpretability regulations. Research was chosen based upon relevancy and diversity of approaches. Keywords such as, interpretability, model-agnostic, machine learning and healthcare were combined in Google Scholar to search for papers.

Literature 2.2.2

Scalability

Scalability describes the ability to efficiently handle massive datasets and complex computations it is a critical factor in realising the potential of ML in various domains. Scalability often necessitates rethinking and redesigning traditional ML algorithms to ensure efficient processing of vast amounts of data. This might involve introducing approximations, distributed computing paradigms, or entirely new algorithms that maintain predictive power while mitigating computational burdens Bekkerman et al (2012).

In Mohammad et al. (2018) the authors aim to provide solutions to scaling ML algorithms to handle real-world issues, in their research they focus on their massive increase of credit card transaction data in relation to fraud detection. They used the algorithms RF, Balanced Bagging Ensemble (BBE) and Gaussian Naïve Bayes to investigate potential for scalability and prediction accuracy on big datasets. A smaller dataset was introduced as a control, for the control BBE outperformed the others but could not extrapolate its performance to the big dataset. The RF algorithm proved most scalable and more robust in handling big datasets.

Liu et al. (2013) investigated the effects of Naïve Bayes Classifier (NBC) in large datasets, to explore the effects of scalability upon the algorithm. The authors concluded that as the dataset increases the accuracy also improves (82% accuracy on largest dataset). The data set used consisted of millions of movie reviews and the algorithm showed minimal computational limitations when scaling up. Elkan. (1997) also supports this claim with his theoretical research citing the scalability benefits of NBC to be attributed to sparse data handling capabilities, minimal parameter tuning and NBC being able to easily support parallelisation regarding distributed computing.

In Pavlov et al. (2002) the researchers explore the process of making SVM more scalable by implementing boosting to the Sequential Minimal Optimisation (SMO) algorithm which is the widely adopted technique for training SVM's. The researchers concluded that the Boost-SMO uses much less memory than the Full-SMO yet also maintaining the same level of accuracy. Similarly, Razzaghi & Safro. (2015) attempt to resolve the memory and computational limitations of training SVM on big datasets. They introduce a multi-level algorithmic framework (MF) inspired by multiscale optimisation strategies. The MF leverages techniques at various levels of algorithm design to achieve significant reductions in training times. By combining data preprocessing, subsampling, parallelisation, model simplification, active learning, and other strategies, the framework addresses

the challenges of scalability and computational efficiency. Through their results they concluded that the MF substantially improves the computational time without losing the quality of classifiers.

In Singh et al. (2014) investigated how compatible ML algorithms were when implemented across parallel networks to allow for more computational capacity. The researchers used Mahout to build distributed ML networks of the ML algorithms, RF, SVM, Naïve Bayes and ANN. These algorithms were trained upon a large, distributed dataset. It was concluded that RF outperformed its counterparts whereas ANN performed the worst suffering a heavy setback with the size of the data.

Literature	Study	Domain	ML Algorithms	Research Method	Key Findings
Scalability	Mohammad et al. (2018)	Credit Card Transactions	RF / BBE / NB	Comparison of ML algorithms to determine scalability potential	RF proved most scalable and robust in handling big datasets
	Liu et al. (2013)	Movie Reviews	NBC	Investigation of how NBC deals with growing datasets	Showed little computational limitation when scaling up
	Pavlov et al. (2002)	Online Analytics Data	SVM	Implementing Boost-SMO to SVM algorithm to analyse scalability benefits	Boost-SMO uses much less memory than Full-SMO whilst maintaining similar levels of accuracy
	Razzaghi & Safro. (2015)	Diverse range of datasets used	SVM	Investigate scalability benefits of MF on SVM	MF substantially improves the computational time without losing the quality of classifiers.
	Singh et al. (2014)	Network / Security Data	RF / SVM / NB / ANN	Investigate the compatibility of distributed ML for ML algorithms	RF outperformed its counterparts whereas ANN performed the worst suffering a heavy setback with the size of the data.

Table of ML scalability studies

Interpretability

Interpretability is an important factor within certain domains such as healthcare, finance and legal systems Adadi & Berrada. (2018). These domains operate on high stakes sensitive data, by providing explanations for predictions, these models empower users, enhance accountability, and ensure that AI systems are used responsibly and ethically. This fosters greater adoption and acceptance of AI technologies in critical applications while minimising potential risks and pitfalls associated with black box models.

In Katuwal & Chen. (2016) the authors emphasise the importance of interpretability within highly complex models. They used a RF algorithm on electronic healthcare records investigating mortality rates in the intensive care unit. They proposed a Model-Agnostic Explanations algorithm (LIME) to explain the predictions made by the RF. Ultimately the researchers were able to accurately predict the mortality rate of ICU patients (80% accuracy) and were also able to uniquely identify the contribution of the important features on mortality prediction for each patient. They verified their results by cross referencing them with existing clinical knowledge and the important features were aligned.

Similarly, Elshawi et al. (2019) demonstrated the utility of various model-agnostic explanation techniques of machine learning models. They used RF for predicting the individuals at risk of developing hypertension. The authors split the model-agnostic techniques into two separate categories, global interpretability (Feature Importance, Partial Dependence Plot (PDP), Individual Conditional Expectation (ICE), Feature Interaction and Global Surrogate Models) and local interpretability (Local Surrogate Models (LIME), Shapley Value Explanations). Then compared the results on the dataset, they concluded that the different interpretability techniques shed light on different aspects, the global interpretations allow a greater understanding of the entire conditional distribution modelled by the trained response function. However, local interpretations promote the understanding of small parts of the conditional distribution for specific instances. The authors emphasise that the use of interpretability techniques need to be context specific for the application need.

Wu et al. (2023) propose to address the issue of needing interpretable ML algorithms within the healthcare industry. They conducted research investigating the key predictor variables involved

within heart disease and diabetes. The authors acknowledge that traditional ML algorithms can be used to identify feature importance but are not as useful for establishing coefficients in relation to ranking feature importance. They propose an approach of deep learning with a local interpretable model–agnostic explanations (LIME)-based interpretable recommendation system to solve this problem. The authors local LIME model was applied to several ML algorithms from complex multi-layer perception algorithms to simpler LR algorithms. The feature importance ranking of the local LIME models seemed to align with every model expect for an exception in the DT model. The authors propose LIME as a method for interpreting complex models using healthcare data.

Wu et al. (2021) used model-agnostic interpretability techniques in conjunction with ML algorithms to identify relevant biomarkers associated with severe infection and increased risk of death on COVID-19 patients. The authors use 4 ML algorithms (DT, RF, Gradient Boosted Trees and NN) after training these models a variety of interpretability techniques were applied to gain insights from the predictions. Permutation feature importance was used to find the most important features. PDP, ICE and ALE to visualise the inter-feature relationships. Finally, LIME and SHAP were used to identify the reasoning for false negatives in the classification. The authors cross referenced their results associated with feature importance with existing medical anecdote and COVID-19 patient data from Brazil the results could be cross-validated.

Karatza et al. (2021) explored the optimisation of interpretability in ML models predicting the onset of breast cancer. Three highly complex models were trained on breast cancer data (RF, NN and an Ensemble of Neural Networks (ENN)). The model-agnostic methods of GS, ICE and SV were applied alongside the ML algorithms to make feature selections and determine feature importance. The authors identified features and symptoms for the early discovery of breast cancer and these findings were validated through medical knowledge and evaluation on datasets.

Literature	Study	Domain	ML Algorithm	Model-Agnostic Algorithm	Research Method	Key findings
Interpretability	Katuwal & Chen. (2016)	Healthcare	RF	LIME	Investigate wheter LIME enhances interpretability of RF algorithm	LIME improves interpretability by identifying contribution of most important features
	Elshawi et al. (2019)	Healthcare	RF	Feature Importance / PDP / ICE / LIME / Global Surrogate Models / SVE	Understand different aspects of interpretability methods	Local and Global interpretations serve different purposes need to be adopted in context specific scenarios
	Wu et al. (2023)	Healthcare	NB / LR / DT	LIME	Investigate the effect of LIME in enhancing interpretability in popular ML algorithms	Feature importance ranking of the local LIME models seemed to align with every model expect for an exception in the DT model.
	Wu et al. (2021)	Healthcare	DT / RF / GBDT / NN	PFI / PDP / ICE / ALE / LIME / SHAP	Investigated using a combination of Model-Agnostic approaches to improve interpretability in COVID-19 model	Each interpretability method served a separate purpose and combined well to improve interpretability
	Karatza et al. (2021)	Healthcare	RF / NN /ENN	GS / ICE / SV	Used Model-Agnostic approaches alongside ML approaches to enhance intepretabiltiy of models by determing feature importance	Interpretability methods identified features and symptoms related to early onset of breast cancer

Table of ML interpretability studies

Critical Analysis 2.2.3

Interpretability Vs Performance Trade-off

Within the literature researchers focus upon the development of more interpretable ML algorithms and models but focus less upon the potential trade-offs associated with the pursuit of interpretability. Fundamentally there is an issue that simpler more interpretable ML algorithms may sacrifice prediction accuracy as compared to deeper more complex algorithms.

Even with the use of model-agnostic approaches elements of performance and complexity are sacrificed in exchange for interpretability. Techniques such as LIME and SHAP work by approximating the behaviour of black-box models with a simpler more interpretable model, such as LR or DT. In the process of this simplification the new model may not capture certain nuances and patterns of the original model leading to a potential loss in performance and complexity.

In order to mitigate the trade-off of performance it may be appropriate to implement a combination of model-agnostic techniques to provide a more comprehensive understanding of the black-box model. Wu et al. (2021) demonstrates this in their research. The ensembling of different interpretation methods can mitigate the risk of bias introduced by a single technique and provide a more robust interpretation.

Additionally, the specific domain the ML algorithm is deployed in also influences the interpretability vs performance trade-off. For instance, there are regulatory bodies within the healthcare industry which ensure that ML models relating to explaining diagnoses and treatment recommendations are interpretable to a certain extent. The Health Insurance Portability and Accountability Act (HIPAA) require that patients have the right to understand how their health data is used, including by ML models HIPAA. (1996). The regulations and precautions associated with ML in the healthcare industry are strict as the consequence for error is relatively high. In comparison, the interpretability demand for an energy consumption ML model is not as high as the consequence of error is not as great. Therefore, the domain is an important feature in determining the extent of interpretability as in the case of energy consumption a greater proportion of interpretability can be sacrificed for performance whereas in the healthcare industry certain interpretability thresholds must be met.

Scalability Vs Performance Trade-off

ML Algorithms designed for scalability may prioritise efficiency and speed at the expense of the model accuracy. For instance, scalable algorithms may assume certain data distribution or data preprocessing steps to achieve efficiency. Deviations from these assumptions can lead to suboptimal performance. In addition, these algorithms may be programmed to operate on limited depth and width due to resource constraints. This can result in a shallow network that struggles to learn complex patterns within the data.

Also, there are limitations associated with dealing with larger datasets. Preprocessing the data can be challenging identifying noise, outliers and missing values may be increasingly difficult as the dimensions of the dataset increase, traditional visualisation methods such as histograms and boxplots may not be a viable option. Datasets with a large proportion of noise and outliers may introduce biases in the predictive ML models once trained. Market sectors such as social media and telecommunications rely heavily on real-time data and are time constrained on the processing time. Hence, computational efficiency and shortened training times are imperative for the sustainability of these systems. To ensure fast and efficient preprocessing and training ML systems adopt parallelisation for a greater processing capacity, this can be beneficial for optimising hyperparameter tuning as well. In addition, reinforcement learning is also incorporated to continuously monitor and evaluate the performance of the models in real time.

Conclusion 2.3

One of the main findings from this literature review is that the implementation of ML algorithms is very context specific. There is a diverse range of industries within the public sector that implement ML methods, the types of algorithms they adopt will differ from industry to industry as algorithms are suited towards different data structures and prediction algorithms. For instance, within the fraud-detection domain the datasets which algorithms are trained on are often highly imbalanced with the target feature occupying less than 1% of the dataset. ML algorithms such as SVMs are better adapted to identify rare events or anomalies making them more suited towards imbalanced data. Conversely, within the healthcare domain the datasets are characteristically more high-dimensional. Therefore, complex ensemble approaches such as RF, gradient boosted algorithms and deep neural networks may be implemented more. There is little consensus about which algorithm is best as ML is a very

multi-faceted practice and is rarely a one size fits all scenario. Therefore, it should be the role of the practitioner to implement an algorithm which best addresses the specific problem. However, looking at the review there is a clear consensus that more complex ensemble approaches seem to outperform singular classical ML algorithms and the popularity of RF seems to be prevalent across multiple domains.

This concept can also be applied within regarding limitations of ML algorithms in the public sector. The limitations of ML algorithms are also very context specific. Within some industries more of an emphasis needs to be placed on scalability due to the volume of data coming through the pipelines. In other industries more of an emphasis needs to be places on interpretability due to the regulations surrounding the domain and the severity of the consequences associated with using ML for decision making. There are trade-offs with performance when regarding scalability and interpretability, so practitioners need to balance off the trade-offs and implement an approach which best suits their needs.

Gaps in Literature 2.4

The literature mainly focuses upon real-world case studies which involve the evaluation of large entities or large-scale ML projects. There is an underrepresented area of literature discussing the use cases for smaller entities who may not possess the resources to operate ML models on a large scale. More sophisticated techniques such as distributed ML may be impractical for some practitioners as they may want to adopt ML on a smaller scale. Presently, ML has become more accessible with open-source software such as Python to allow individuals with little expertise to train their own ML models. Therefore, it would be valuable for research to be done discussing the implementation of ML models using accessible software such as python taking into consideration CPU and GPU constraints. As this will act as a framework for practitioners to pursue training ML models.

Reviewing the literature on this topic it is evident that some domains have been early adopters of ML (industries within the public sector discussed earlier within the review). However, it would be beneficial for more research related to the adoption of ML algorithms in less common industries for analytical predictive modelling e.g., education and agriculture. The implementation of ML within education can be hugely beneficial for both students and teachers. For students the implementation of personalised learning features using ML can enhance development and for teachers ML can be implemented to assist in assessment and grading of work to save time. As for agriculture, predictive modelling using ML can improve the prediction of crop yields based on various factors such as weather data, soil conditions, and crop management practices. Also, ML can be implemented to detect crop diseases and pest infestations early. Sectors such as agriculture and education wouldn't have been traditionally believed to be assisted with AI technology. But this makes the prospect of ML within these industries even more enticing, this is because there is so much untapped potential the implementation of using ML to assist with operations has the prospect to produce revolutionary results.

Methodology (Chapter 3)

This research aims to investigate the performance of traditional ML algorithms (LR, DT and NN) with more complex ensemble approaches (BR, CatBoost, LightGMB and RF) to identify if more complex ML algorithms perform better. In addition, this research will analyse various feature selection methods to identify the effectiveness of feature selection methods and control for feature selection bias when evaluating the algorithms, Guo et al. (2018) adopted this method in their research with the rationale of controlling for dataset bias. The feature selection approaches were deployed on the full data creating several subsamples, after which the ML algorithms were trained on each separate

subsample and the results detailing the performance metrics of each algorithm are consolidated in a table within the results section. Finally, the computational demands will also be considered and factored in with the performance of the algorithm to present a more comprehensive well-rounded evaluation of the ML algorithms.

LR, DT and NN were chosen as the singular ML algorithms because regarding the literature they are arguably the most cited and compared ML algorithms that are used within real-world applications. Each algorithm has practical applications in different domains. LR is commonly used in economics and social sciences. DTs are popular in fields like finance and healthcare. NN excel in tasks like computer vision, natural language processing, and deep learning. Additionally, each ML algorithm represent different classes of machine learning algorithms, each with its own modelling philosophy and assumptions. This diversity allows for a wide evaluation of modelling strategies.

BR, CatBoost, LightGBM and RF were chosen for the ensemble approaches as they are all tree-based algorithms. The reasoning behind only picking tree-based approaches was because of time / computational constraints. Tree-based algorithms can handle high-dimensional feature spaces efficiently, this is done by selecting and splitting only on relevant features, making them suitable for datasets with many features by heavily reducing training times Aluja-Banet & Nafria. (2003). Also, when dealing with sparse data tree-based algorithms can perform efficiently because they only need to make splits based on the available features with non-zero values. Additionally, BR and RF use a bootstrap aggregating techniques whilst LightGBM and CatBoost use a gradient boosting technique so employing a variation on techniques may produce more diverse and compelling results. Also, LightGBM and CatBoost can handle categorical data without the need of encoding into dummy variables therefore it will be interesting to see how this affects performance.

The research data used was a subset taken from the official NHS database, the intention of the dataset is to predictive model the total cost of individuals to allocate funding to NHS branches within the UK. Hence the total cost variable was taken as the target variable and all other variables were treated as predictor variables.

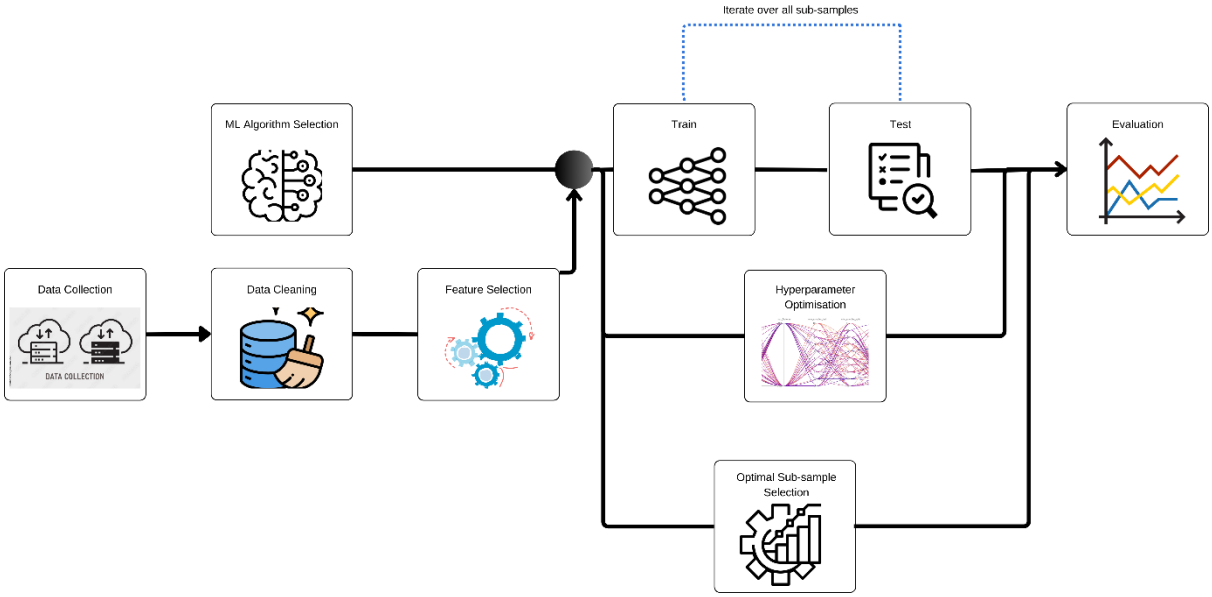


Diagram of Methodology

Data Preparation and Cleaning 3.1

(The red text referencing the corresponding figure in the appendix details the Python code related to coding the appropriate data preparation/cleaning task).

Merging 3.1.1

The first task was to merge the multiple datasets together (Figure 1.1), this is because it would be desirable to train models on a larger dataset because as sample size increases the variance decreases Zhou et al. (2017). It will also allow a larger split of training and validation data to ensure the models are not overfit. All ML algorithms were trained on a 50:50 train, test split to ensure a fair comparison. The data merging, cleaning and exploration was conducted within Python primarily using the Pandas and NumPy libraries.

Missing Values 3.1.2

After merging the final data frame was comprised of 50,000 rows by 1,279 columns. Using the is.null() function it stated that there was 172,504 missing values within the data frame (Figure 2.1). Missing values can limit model performance in ML by introducing biases and distorting relationships between variables, therefore it is critical to remove or impute missing values Masconi et al. (2015). Four specific columns were identified with a large proportion of missing values ('eligibleGP', 'rgp', 'selGP', 'ind_ethnicity') 98.2% of the missing values occurred within these specific columns so it makes sense to just remove the columns rather than impute values (Figure 2.2 / 2.3). After deleting the columns, the .drop() function was used to remove all rows containing residual missing values (Figure 2.4). After this the data frame had 0 missing values with the dimension of 49,929 rows and 1,275 columns (Figure 2.5). The dataset is very sparse, 87.05% '0' values (Figure 2.6). This is due to a high proportion of binary dummy variables. 1,028 / 1,275 columns are binary dummy variables (Figure 2.7).

Outlier Removal 3.1.3

Considering that the target variable was the total cost, it would be useful to explore the frequency distribution of this variable. A histogram was plotted with the 'total cost' variable as seen from the histogram a large proportion of individuals are lower in total cost with very few individuals on the higher end (Figure 3.1). In fact, 28,902 individuals have a zero total cost (Figure 3.2) which is approximately half of the sample size. A boxplot was plotted to visualise outliers, from the boxplot (Figure 3.3) there are a high proportion of outliers even 2 cases with over £80,000 in total cost, these outliers need to be removed as they can heavily influence the sensitivity of the ML model Nyitari & Virag. (2018). For the removal threshold of outliers 1.5 times the interquartile range (IQR) plus the 75^{th} percentile is going to be used. Hence, 369(1.5*IQR) + 246(75^{th} percentile) = 615 (Figure 3.4), so all values over 615 are to be removed. There are 7,592 values over the total cost of 615 (Figure 3.5), so 42,337 rows of data will remain after outliers are removed (Figure 3.6).

Normalising of Variables 3.1.4

All numerical columns were normalised (categorical columns were identified and excluded from normalisation) so that all values were between [0,1] (Figure 4.1) this is because many ML algorithms are sensitive to the scale on input features. When features have different scales, some algorithms may give more importance to features with larger scales leading to biased results Borkin et al. (2019). An additional benefit to this is that normalisation of values can often contribute to faster convergence speeds.

Converting Categorical Values to Dummies 3.1.5

Also, all categorical variables were converted into dummy variables (Figure 5.1). The reasoning for this was majority of ML algorithms require numerical input data (LR, BR, RF, NN etc). Also, it prevents

assumptions about the magnitude or interval between categories Gnat. (2021). For instance, within the data household type was a variable. A ML algorithm may interpret two-parent to be an order of magnitude higher than single-parent this could lead to misleading results when training. However, some ML algorithms such as LightGBM and CatBoost have been designed to be able to handle categorical features so the categorical features were left in when training on these algorithms.

Constant Features 3.1.6

Constant features were also removed as they did not provide any useful information for distinguishing across samples (Figure 6.1). Constant features have zero variance because all data points have the same value. This can lead to inefficiency in model training and can even cause some algorithms to malfunction or produce unstable results. Also, constant features occupy memory and computational resources without adding any value. This can slow down the training process and increase the memory requirements of the algorithm.

Feature Selection 3.2

Due to the sparsity of the data, it made sense to reduce the dataset via selecting the features which were most significant in relation to the target variable of total cost. This would reduce overfitting potential and enhance robustness of the model by mitigating the 'curse of dimensionality' Bellman. (1957). In addition to this, it will also allow for more methods of training ML models as through dimensionality reduction it will mean data processing capacity of certain models is no longer a limitation.

For the feature selection process, it was decided to adopt several methods to create a variety of sub-samples of the data. The reason for this was to limit bias through mitigating the concept that a specific feature selection method would cater more to a specific ML model. Through this method model performance can be judged across an array of sub-samples and outliers of model performance can be identified easily. The feature selection methods chosen were Elastic Net Regularisation (ENR), Gradient Boosted Decision Trees (GB), K-Best, Lasso Regularisation, Random Forest (RF) and Variance Threshold (VT). RF and GB are ensemble algorithms so it will also be interesting to compare the performance of ensemble algorithms with singular algorithms for feature selection.

Multicollinearity 3.3

Multicollinearity is the phenomenon where two or more predictor variables in a predictive model are highly correlated, making it difficult to determine the individual effect of each variable on the target variable. In this research multicollinearity is not directly addressed by using VIF to identify highly correlated variables. This is because as mentioned before the dataset contains a large proportion of binary variables, binary variables take on only two values (0 or 1). This limited range of values reduces the potential for high variability between them, making it less likely for two binary variables to be strongly correlated. Binary variables are inherently orthogonal (uncorrelated) to each other. If two binary features are independent, they will not be collinear. This property helps mitigate the issue of multicollinearity. In addition to this the use of regularisation techniques (Lasso & Ridge) penalise the selection of highly correlated variables so the prevalence of multicollinearity is also controlled for within the feature selection methods and some of the ML algorithms which utilise boosting.

Feature Selection Process 3.4

(The red text referencing the corresponding figure in the appendix details the python code related to coding the specific feature selection algorithm).

Gradient Boosted Decision Tree 3.4.1

(Figure 7.1) GB is an ensemble learning method that builds multiple decision trees sequentially, each correcting the errors of the previous tree. During the training process as each tree is constructed, it evaluates the importance of each feature in predicting the target variable. Features that help reduce the errors more effectively are assigned higher importance scores Xu et al. (2014). After training the entire ensemble, feature importance scores from individual trees are aggregated to provide a consolidated importance score for each feature. This can be achieved by averaging or summing the importance scores across all trees. The features are then ranked and displayed by importance. Three subsamples were created using GB the top 30 (Figure 7.2), 50 (Figure 7.3) and 70 (Figure 7.4) ranked importance variables were selected for separate sub-samples this is because some ML algorithms may be able to capture deeper patterns and relationships without overfitting. (Top selected features can be viewed at (Figure 7.5)).

Random Forest 3.4.2

(Figure 8.1) Likewise to GB, RF combines the predictions of multiple DTs to perform feature selection. Each tree is built randomly selecting subsets of the training data (bootstrapping) and random subsets of features for each split. The concept of bootstrapping involves creating multiple datasets by sampling with replacement from the original training data. Each dataset is used to train a separate DT. For each split in a DT, RF randomly selects a subset of features from the available features. This randomness ensures that each tree learns different aspects of the data Murphy & Moore. (2019). Each DT is constructed by recursively partitioning the data into subsets based on feature values. The goal of each split is to minimise impurity (Gini index) Sazonau. (2012). Once all trees are built, predictions are made by aggregating the predictions of individual trees. Features that cause the largest decrease in ensemble performance when permuted are considered the most important. The importance scores are calculated as the average decrease in Gini impurity across all trees for each feature. The calculated importance scores allow for the ranking of features based on their contribution to the ensemble's predictive power. Features with higher importance scores are considered more influential. Similarly, to GBR three subsamples were created with the top 30 (Figure 8.2), 50 (Figure 8.3) and 70 (Figure 8.4) ranked importance variables. (Top selected features can be viewed at (Figure 8.5))

Lasso Regularisation 3.4.3

(Figure 9.1) Lasso Regularisation adds a penalty term to the linear regression cost function to encourage the model to select a subset of important features while shrinking the coefficients of less important features towards zero. The regularisation parameter within the equation is denoted ‘α’, as ‘α’ increases the impact of the penalty term becomes stronger hence forcing a greater proportion of coefficients to shrink to zero Torstensson. (2017). After running Lasso Regularisation all the variables that are > 0 are recalled and these are the variables used within the sub-sample. The alpha term was set to 1.5 and at this strength 114 features remained with a non-zero coefficient (Figure 9.2).

Elastic Net Regularisation 3.4.4

(Figure 10.1) EN combines Lasso and Ridge regularisation to achieve a balance between feature selection and parameter shrinkage. Similarly, to Lasso EN also has a ‘α’ penalty term but in addition to this also has an ‘β’ penalty term for ridge regularisation. The difference between Lasso and Ridge is that Ridge doesn’t shrink coefficients to exactly zero as it adds a penalty term to the cost function that is proportional to the square of the coefficients Zou & Hastie. (2005). Combining Lasso and Ridge in ENR strikes a balance between the bias and variance of the model. It maintains a degree of bias reduction like Ridge while still performing feature selection like Lasso. This can lead to feature selection that captures the important relationships while reducing the impact of multicollinearity-

induced instability. For the ENR feature selection alpha was set to 5 and the ratio was set to 50:50 indicating an equal mix of Lasso and Ridge regularisation. After feature selection 149 variables remained (Figure 10.2).

K-Best 3.4.5

(Figure 11.1) K-Best algorithm involves selecting the top 'K' features from a given set of features based on their scores and relevance to the target variable. Each feature is assigned a score based upon the ANOVA F-value relationship with the target variable all variables are then ranked in descending order. Then only top 'K' features are selected for the sub-sample. 'K' was set to 30 (Figure 11.2), 50 (Figure 11.3) and 70 (Figure 11.4).

Variance Threshold 3.4.6

(Figure 12.1) The concept behind the variance threshold approach is to remove features that have low variance across the dataset. The variance of each feature is calculated, variance is a measure of how much the values of a feature spread out from the mean. Features with low variance have little variability and may be less informative. An appropriate variance threshold is set and features below this threshold are removed from the dataset. The variance threshold was set to 0.05 so all features below this threshold were removed, 47 features remained to comprise of the subsample (Figure 12.2).

Individual ML algorithms 3.5

(The red text referencing the corresponding figure in the appendix details the python code related to coding the specific ML algorithm).

Linear Regression 3.5.1

(Figure 13.1) LR is a ML algorithm that assumes a linear relationship between the input features and the target variable. During training the model adjusts the coefficients to minimise the difference between the predicted and actual target values, during training the loss function was set to minimise for Mean Absolute Error (MAE). One of the key benefits of LR is, because of its assumption of linearity, the coefficients of the regression equation can be interpreted to understand the relationship between the input features and the target variable.

Neural Network Regression 3.5.2

(Figure 14.1 / 14.2) NN is based upon the principals of Artificial Neural Networks, which are comprised of interconnected nodes (neurons) that process information in a manner inspired by the human brain. The input layer of the neural network matches the number of input features. Each neuron in the input layer corresponds to one feature. Hidden layers process the input data through a set of weighted connections and activation functions Dinov. (2018). The number of hidden layers and the number of neurons in each layer are hyperparameters that can be tuned. Activation functions introduce non-linearity to the model. The chosen activation function for training was ReLU (Rectified Linear Activation). The activation function determines whether a neuron should "fire" or not based on its input. The output layer produces the final regression prediction.

Decision Tree 3.5.3

(Figure 15.1 / 15.2 / 15.3) A DT builds a tree-like structure to make decisions based on input features and their relationships. The DT building process starts at the root node, representing the entire dataset. At each node, the algorithm selects the best feature to split the data into two or more subsets. This was done based on Gini impurity, which measures the impurity in the data. The algorithm evaluates different features and their values to determine the best split. The goal is to minimise impurity, meaning that the resulting subsets are as homogenous as possible in terms of the

target variable. The tree-building process continues recursively for each subset, creating child nodes for each branch. The algorithm repeats the process of selecting the best feature and split until one of the stopping criteria is met, such as a maximum tree depth, a minimum number of samples per leaf node, or when further splitting doesn't significantly improve impurity reduction.

Ensemble ML Algorithms 3.6

Bagging Regressor (Bootstrap Aggregating) 3.6.1

(Figure 16.1 / 16.2) BR is an ensemble ML algorithm that aims to improve the stability and performance of a regression model by combining the predictions of multiple base models. The primary concept behind bagging is to train multiple instances of the same regression algorithm on different subsets of the training data and then aggregate them together to make a final prediction. For each base model, a bootstrap sample is created from the training set. Bootstrap sampling involves randomly selecting data points from the training set with replacement Khiari et al. (2019). This means that some data points may be selected multiple times, while others might not be selected at all. Each bootstrap sample becomes the training data for one base model, for every base model a different bootstrap sample is used. To make a prediction using the BR, an input instance must be provided. The input instance is passed to each of the trained base models, and each model produces a prediction. The final prediction is calculated by aggregating the predictions from all base models. In the case of regression, this aggregation is done by taking the average of the predictions from all base models.

CatBoost Regressor 3.6.2

(Figure 17.1 / 17.2) CatBoost is a gradient boosted algorithm specifically designed to handle categorical features without the need for preprocessing. Like all boosted algorithms CatBoost focuses upon improving weak learners. It uses the ordered boosting strategy which sorts categorical features and finds the best split points on these ordered values Hancock & Khoshgoftaar. (2020). Also, it provides built-in regularisation techniques (Lasso and Ridge), and random strength to prevent overfitting. These techniques control the complexity of the learned model. Also, the depth of trees can be controlled which can help balance model complexity and predictive power.

LightGBM (Light Gradient Boosted Machine) 3.6.3

(Figure 18.1 / 18.2) Likewise, to traditional gradient boosting LightGBM follows a framework where the goal is to iteratively improve a weak learner of DT. LightGBM uses a leaf-wise growth strategy. It selects the leaf with the maximum delta loss (improvement in the loss function) to grow the tree. This strategy prioritises growing leaves that lead to greater improvements, making the tree more efficient and potentially reducing overfitting Yan et al. (2021). LightGBM uses a gradient-based approach to update the model in each boosting iteration. It calculates the gradients of the loss function with respect to the predictions. It uses these gradients to determine the direction and magnitude of adjustments to the predictions in order to minimise the loss. LightGBM employs a unique technique called 'Gradient-based One-Side Sampling (GOSS)' to reduce memory usage and speed up training Ke et al. (2017). It only considers the data points with large gradients when finding the best feature split during tree growth, discarding less informative samples. Also, it employs regularisation techniques like Lasso and Ridge to prevent overfitting. LightGBM can efficiently handles categorical features without the need for creating dummy variables. It uses techniques like CatBoost to find the best way to split categorical variables.

Random Forest 3.6.4

(Figure 19.1 / 19.2) (Description of RF at Chapter 3.4.2)

Hyperparameter Tuning 3.7

Hyperparameter tuning is necessary as a control when comparing different ML algorithms as it helps ensure that each algorithm is evaluated under fair and optimal conditions. Some ML algorithms have various hyperparameters which can significantly influence the training of the model. Ensuring effort has been taken to identify near optimal hyperparameter values will provide a better insight into comparing the capabilities of the algorithms.

Grid Search is a technique used for hyperparameter tuning in ML models. Grid search exhaustively searches through the provided parameter grid, training and evaluating models for all combinations provided. This helps find the best hyperparameters to optimise the performance of the model based on the chosen scoring metric.

The procedure for defining the hyperparameters for the grid search was not conducted in a systematic way. Instead, the hyperparameters were firstly defined quite conservatively with a large range between them and after training and analysing optimal hyperparameters they were iteratively refined to more of narrowed down range. Also, this method was used because there were time limitations as too extensive hyperparameter searches would be infeasible as the training times of some algorithms were very long.

To view the hyperparameters included within the grid search:

- Neural Network Regression (Figure 20.1)
- Decision Tree (Figure 20.2)
- Bagging Regressor (Figure 20.3)
- CatBoost Regressor (Figure 20.4)
- LightGBM (Figure 20.5)
- Random Forest (Figure 20.6)

Evaluation Metrics 3.8

For evaluation metrics MAE and R^2 were selected as combined they can provide a well-rounded assessment of the performance of the ML models. They offer complementary insights MAE assess prediction accuracy whilst R^2 evaluates the model's ability to capture data patterns. Additionally, both metrics are very easily interpretable which is beneficial when comparing the performance of algorithms.

Mean Absolute Error 3.8.1

MAE measures the average absolute error between the predicted values and the actual values. A lower MAE indicates that the model's predictions are closer to the actual values on average, while a higher MAE suggests that the model's predictions are less accurate. MAE was chosen over other evaluation metrics such as MSE and RMSE as these metrics will penalise larger errors more significantly and gives more weight to the impact of outliers whilst MAE treats errors equally. Also, MAE is more interpretable as the MAE is interpreted in the same unit as the target variable (total cost), which would be £. This is why MAE is frequently used as an evaluation metric for models which involve financial information. Another benefit is that this means the results are interpretable to a non-technical audience.

R-Squared (R^2) 3.8.2

R^2 is used to represent the coefficient of determination, this provides valuable insights into how well a regression model fits the observed data. R^2 measures the proportion of the variance in the target variable that is explained by the predictor variables in the model. Higher R^2 values indicate a better

fit, while lower values suggest that the model does not capture much of the variance. R^2 can only be within the range of 0 – 1 and is a useful tool to compare different models.

Results 3.9

Linear Regression 3.9.1

Linear Regression		
Sub-Sample	MAE	R^2
Full data	>150	<2.5%
Elastic Net	>150	<2.5%
GradientBoosted (30)	94.97	6.27%
GradientBoosted (50)	94.78	6.51%
GradientBoosted(70)	>150	<2.5%
K-Best(30)	>150	<2.5%
K-Best(50)	>150	<2.5%
K-Best(70)	>150	<2.5%
LassoRegularisation	>150	<2.5%
RandomForest(30)	96.41	4.20%
RandomForest(50)	95.93	5.01%
RandomForest(70)	95.84	5.07%
VarianceThreshold	95.67	5.31%

Decision Tree 3.9.2

Decision Tree			
Sub-Sample	MAE	R^2	Optimal Max Depth
Full Data	93.21	7.02%	4
ElasticNet	95.6	5.24%	3
GradientBoosted(30)	91.67	7.58%	4
GradientBoosted(50)	91.95	7.00%	4
GradientBoosted(70)	92.39	6.96%	3
K-Best(30)	96.19	4.51%	3
K-Best(50)	95.53	5.27%	3
K-Best(70)	95.53	5.27%	3
LassoRegularisation	92.39	6.96%	3
RandomForest(30)	92.91	6.95%	3
RandomForest(50)	92.13	7.28%	3
RandomForest(70)	92.13	7.28%	3
Variance Threshold	92.07	7.30%	4

Neural Network Regression 3.9.3

Neural Network			
Sub-Sample	MAE	R^2	Optimal Hidden Layer
Full Data	>150	<2.5%	32
ElasticNet	94.97	5.51%	16
GradientBoosted(30)	91.01	8.83%	64
GradientBoosted(50)	91.42	8.52%	32
GradientBoosted(70)	92.91	7.76%	32
K-Best(30)	96.26	4.28%	8
K-Best(50)	95.47	5.01%	8
K-Best(70)	95.61	5.35%	8
LassoRegularisation	93.18	6.02%	32
RandomForest(30)	94.16	5.56%	64
RandomForest(50)	94.94	5.92%	16
RandomForest(70)	94.67	6.06%	32
Variance Threshold	94.15	6.87%	64

Bagging Regressor 3.9.4

Bagging Regressor				
Sub-Sample	MAE	R^2	N_estimators	Max_Samples
Full Data	90.34	7.32%	500	0.8
ElasticNet	95.56	3.89%	200	1
GradientBoosted(30)	92.14	6.98%	500	0.8
GradientBoosted(50)	92.72	6.67%	500	1
GradientBoosted(70)	90.33	7.50%	500	0.8
K-Best(30)	94.02	4.11%	200	1
K-Best(50)	93.38	4.23%	500	1
K-Best(70)	95.72	3.89%	500	1
LassoRegularisation	92.18	5.25%	500	1
RandomForest(30)	94.63	3.87%	500	1
RandomForest(50)	94.93	4.74%	500	1
RandomForest(70)	94.09	4.72%	500	0.8
Variance Threshold	92.32	4.97%	200	1

Catboost Regressor 3.9.5

Catboost Regressor					
Sub-Sample	MAE	R^2	Optimal Depth	Optimal Learning Rate	N_estimators
Full Data	90.07	9.35%	4	0.01	500
ElasticNet	93.74	6.55%	5	0.0075	500
GradientBoosted(30)	90.07	9.20%	2	0.01	500
GradientBoosted(50)	89.9	9.51%	3	0.01	500
GradientBoosted(70)	89.79	9.72%	4	0.01	500
K-Best(30)	95.13	4.96%	2	0.0075	250
K-Best(50)	94.29	5.78%	3	0.075	500
K-Best(70)	94.38	5.80%	3	0.075	500
LassoRegularisation	90.25	8.94%	5	0.01	250
RandomForest(30)	90.12	8.72%	2	0.01	500
RandomForest(50)	90.43	8.80%	3	0.01	500
RandomForest(70)	90.54	8.76%	2	0.01	500
Variance Threshold	92.1	7.11%	3	0.075	500

LightGBM 3.9.6

LightGBM						
Sub-Sample	MAE	R^2	Optimal Max Depth	Optimal Learning Rate	Number of Leaves	N_estimators
Full Data	87	15.71%	5	0.01	31	500
ElasticNet	90.26	13.07%	10	0.01	31	250
GradientBoosted(30)	88.1	13.00%	5	0.01	63	500
GradientBoosted(50)	87.61	14.30%	5	0.01	63	500
GradientBoosted(70)	86.01	17.70%	15	0.01	31	500
K-Best(30)	92.63	10.99%	5	0.01	31	500
K-Best(50)	93.19	10.92%	10	0.01	31	250
K-Best(70)	92.98	11.32%	15	0.01	63	500
LassoRegularisation	91.23	11.75%	5	0.01	31	500
RandomForest(30)	89.02	12.98%	10	0.01	63	500
RandomForest(50)	87.98	14.38%	5	0.01	90	500
RandomForest(70)	89.04	12.40%	5	0.01	63	250
Variance Threshold	90.12	12.12%	5	0.01	63	500

Random Forest 3.9.7

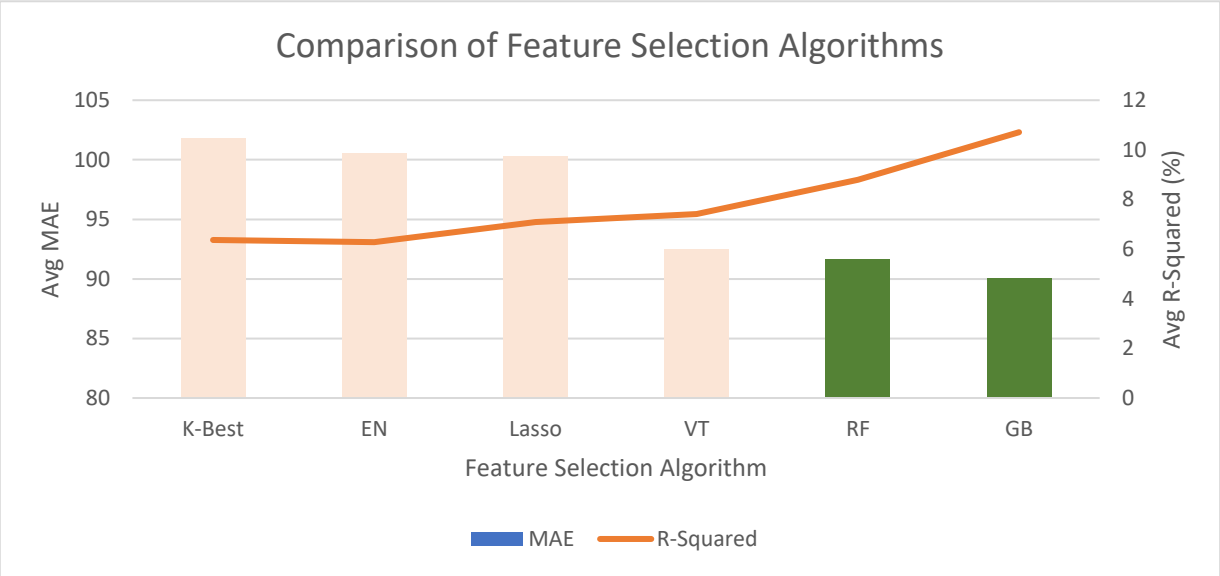
Random Forest				
Sub-Sample	MAE	R^2	Optimal Max Depth	N_estimators
Full Data	87.21	15.21%	10	500
ElasticNet	91.08	7.21%	15	200
GradientBoosted(30)	87.7	17.01%	10	500
GradientBoosted(50)	88.13	14.18%	15	500
GradientBoosted(70)	86.72	17.12%	10	500
K-Best(30)	92.84	7.92%	10	500
K-Best(50)	91.26	10.12%	10	500
K-Best(70)	93.1	7.70%	10	500
LassoRegularisation	92.61	8.18%	5	500
RandomForest(30)	88.04	13.12%	10	500
RandomForest(50)	87.11	15.19%	15	500
RandomForest(70)	87.83	12.96%	15	500
Variance Threshold	91.01	8.19%	5	200

Computational Capacity Comparison 3.9.8

Computational Requirements Ranking		
Algorithm	Rating	Notes
Linear Regression	10	<30 secs for all subsamples, <1 min for full data
Decision Tree	9	<1 min for subsamples, <2mins for full data
Neural Network	7	<2mins for subsamples, <10mins for full data
Bagging Regressor	2	<1.5hrs for subsamples, <2hrs for full data
Catboost Regressor	5	<10mins for subsample, <20mins for full data
LightGBM	4	<15mins for subsample, <25mins for full data
Random Forest	2	<1.5hrs for subsamples, <2hrs for full data

Analysis and Discussions (Chapter 4)

Feature Selection Analysis 4.1



(The bars in green represent ensemble methods whilst the bars in light orange represent singular methods)

Referring to the chart comparing feature selection algorithms, on average the GB algorithm outperformed all others, with RF coming in second and VT third. EN, K-Best and Lasso did not perform as well but in this case, there were a few outliers as these feature selection methods tended to overfit with LR, meaning their mean MAE and R^2 were slightly skewed hence it should be noted that the disparity between these algorithms may be overrepresented. GB was by far the best as its subsample's had the lowest MAE and highest R-squared on 7/7 algorithms.

Effects of Subsample Size on Algorithms 4.1.1

Interestingly, all ensemble algorithms were best trained on the larger 70 feature GB sample. Whilst the LR algorithm trained the best on the 50 feature GB sample and NN / DT were trained best on the 30 feature GB sample. This is most likely a result of ensemble algorithms being able to capture more complex relationships and patterns present within higher dimensional complex datasets as the combination of multiple models mitigates the risk of overfitting and enhances the robustness of the predictions. In addition, the performance of ensemble algorithms on the full dataset was much better than that on NN and LR as the full dataset was too complex for these algorithms resulting in overfitting. However, as the largest GB sample only had 70 features it may be the case that the ensemble ML algorithms would have performed even better on a dataset with more features.

Related Research 4.1.2

The result of the research is backed by Xu et al. (2014) who also investigated the capabilities of GB algorithm for feature selection. Likewise in their research GB outperformed RF and Lasso for feature selection. The authors cite the GB algorithm's ability for identifying nonlinear feature interactions as a key component in the effectiveness of the algorithm. Especially in real world cases the relationship between the target and predictor variables is very rarely linear so GB's capabilities to use DTs, which can create nonlinear partitions of the feature space, meaning nonlinear interactions can be captured effectively. In addition, GB use of DTs for feature selection means it's well suited for sparse datasets as DTs can naturally handle large proportions of '0' values very effectively. The authors also reference GB potential for scalability alluding to its efficient sequential tree building structure

focusing mainly upon high importance variables as well as its histogram-based methods that are well-suited for large datasets.

Likewise, Canedo & Betanzos. (2019) also emphasised the potential of ensemble ML algorithms in feature selection. Citing that they strike the balance between producing diverse results as well as maintaining stability so that the features selected are robust. They also state that using ensemble ML for feature selection is easier to optimise referencing research which employed forward and backward search strategies to design a function that could reflect both accuracy and diversity which gave insight into the number of ensembles needed to achieve the highest accuracy.

Feature Interactions and Nonlinearity 4.1.3

K-Best, EN, Lasso and VT all do not consider feature interactions within their algorithms. Feature interactions is a critical aspect of feature selection as complex relationships between features need to be captured to identify intricate patterns within the data which can lead to more generalisable and robust predictions. Additionally, Lasso and EN have a linearity assumption between feature and target variable this may mean nonlinear informative features are not captured correctly. K-Best and VT also don't account for nonlinear relationships as they strictly focus on individual variables.

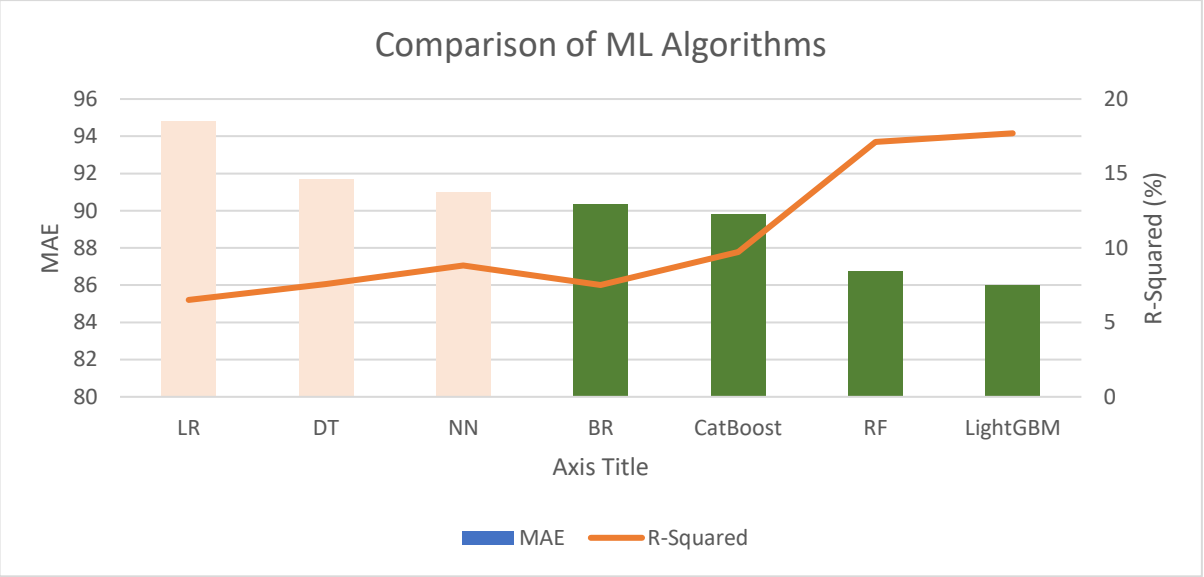
Sparsity 4.1.4

K-Best, EN, Lasso and VT are all not suited for feature selection on sparse datasets. K-Best and VT does not perform well on sparse data as the ANOVA F-value and variance score may be biased against sparse features and these low variance features may be deemed as unimportant due to their sparsity but may in fact contribute to the target variable. Similarly, regularisation within EN and Lasso may be bias towards sparse variables and shrink these down to 0 coefficient when they may contribute significantly.

Singular models 4.1.5

K-Best, Lasso, EN and VT are all singular models, meaning they are all standalone methods as compared to GB and RF which use the combination of multiple feature selection models. Singular algorithms may have less robustness to data variability as they can be more sensitive to variations in the dataset as with ensemble approaches the combination of multiple models can smooth out the variability. This same concept can also be applied to producing more reliable rankings of feature importance as the ensemble of multiple models can offer more robust predictions on feature importance as compared to singular models.

ML Algorithm Analysis 4.2



(The bars in green represent ensemble ML algorithms whilst the bars in light orange represent singular ML algorithms)

The LightGBM algorithm was superior regarding performance. It had the lowest MAE of 86.01 and the largest R^2 of 17.70%. Closely in second place was RF with an MAE of 86.72 and an R^2 of 17.12%. These algorithms performed significantly better than the others. Also, from the results it can also be concluded that all ensemble algorithms performed better than singular algorithms. Seemingly verifying the original hypothesis of the performance of ensemble algorithms being superior. LR performed significantly worse in comparison to others often overfitting under certain subsamples it managed a MAE of 94.78 and only an R^2 of 6.51%. DT and NN were relatively close to the performance of BR and CatBoost with DT and NN having MAE and R^2 scores of 91.67 / 7.58% and 91.01 / 8.83% respectively. Compared to BR and CatBoost having MAE and R^2 scores of 90.33 / 7.50% and 89.79 / 9.72% respectively.

Related Research 4.2.1

The existing literature regarding the comparison of ML algorithms generally favours ensemble approaches performing better. The RF algorithm has had extensive research done within several domains within the public sector ranging from fraud detection to healthcare across different dataset types and circumstances. The consensus of RF as a high performing algorithm is very evident within the literature in addition the use of it in various domains also demonstrates the versatility of the algorithm (Khare et al. (2018), Patil et al. (2018), Pham et al. (2020), Wang et al. (2018)). From the literature in comparing algorithms LR is often overshadowed by its ensemble counterparts (Sahin et al. (2011), Khare et al. (2018)). There has been a tendency for overfitting associated with the LR algorithm and this has been substantiated by the results of this research. Additionally, this overfitting is hard to mitigate due to the lack of hyperparameters associated with LR so therefore it is very difficult to iteratively improve the LR model mainly improvements are confined to optimising the features within the subsample.

BR has been well discussed within the literature frequently popping up in research using real-world case studies in domains such as finance, healthcare and marketing (Yan et al. (2009), Elgimati. (2020)). In comparison studies this ensemble algorithm typically significantly outperforms singular algorithms. However, within this research the performance of BR wasn't that much better as compared to singular algorithms such as NN and DT. This can be explained by a multitude of factors however it is not simple to identify one main cause. Firstly, BR the size and relatively sparse nature of

the dataset may be a reason for why BR did not perform well training. As the number of features in a dataset increases, the volume of the feature space grows exponentially Canedo et al. (2016). In high-dimensional spaces, data points tend to become sparser, making it challenging to obtain reliable and diverse bootstrap samples. The diversity of the samples is essential for bagging to be effective. When data points are sparse, the bootstrap samples may not capture enough variations in the data. Also, the large computational requirements meant that less hyperparameters were used within the grid search (discussed in greater detail in Chapter 4.5).

High Dimensionality and Sparsity 4.2.2

On the contrary, LightGBM and RF seemed to handle the high-dimensionality and sparsity of the data very effectively. LightGBM employs a histogram-based learning algorithm rather than a traditional depth-first tree learning approach. In this histogram-based approach, the data is binned into discrete bins (histograms), and are used to build trees. This method is highly efficient with sparse data because it can skip empty bins, reducing computation time and memory usage Zhang et al. (2019). Also due to its lead-wise growth strategy it selects the lead node that maximises the reduction in loss function. This is suited for sparse datasets as it can quickly identify important splits, further improving efficiency. LightGBM's memory-efficient data storage format, which uses a columnar structure, is advantageous when working with binary data. This format allows for efficient access to specific features while reducing memory consumption Ke et al. (2017).

As for RF, the algorithm has a special feature which introduces randomness during the training process by considering only a random subset of features at each split point in each DT. This feature randomisation reduces the influence of irrelevant or noisy features, making the algorithm robust to high-dimensional data with many irrelevant features Murphy & Moore. (2019). RF also creates multiple subsets of the training data. This sampling technique introduces diversity into the individual DT, which helps the ensemble generalise well to high-dimensional data with sparse patterns Sazonau. (2012). It also reduces the impact of outliers. Additionally, RF provides an OOB error estimation during training, which acts as an internal validation set. This estimation helps assess model performance without the need for an additional validation dataset, making it useful for sparse datasets where processing a large amount of data may be challenging.

Also, RF training times were one of the worst out of all algorithms this is because each base model is trained independently on a different subset of the data. The computational demands increase linearly with the number of base models. However, this may be a result of the hyperparameters chosen within the grid search.

Hyperparameter Analysis 4.2.3

When looking at the results the algorithm with least hyperparameter tuning potential was LR and this was also the worst performing algorithm. Whilst the best performing algorithm, LightGBM, underwent the most extensive hyperparameter tuning. Throughout all different algorithms there is correlation between potential for hyperparameter tuning and performance suggesting hyperparameter tuning is crucial in optimising algorithm performance.

Optimal Max Depth

All ensemble ML algorithms used were tree-based models so it will be interesting to compare the optimal max depth of DTs used within the ensemble and also see how this compares with the singular DT algorithm. For the sigular DT the optimal max depth was 4 and across all other subsamples the average optimal max depth across all subsamples was 3.31. For the CatBoost Regressor the optimal max depth was also 4 and the average was 3.15 interestingly lower than that of the singular DT algorithm. LightGBM had an optimal max depth of 15 and an average of 7.69. RF

had an optimal max depth of 10 and an average of 10.77. For BR the max depth was standardised to 5 as there was no hyper parameter tuning available for max depth in Python instead all models were standarised to 5.

The larger optimal max depth in the LightGBM and RF models demostrate the models superiority as these were also the highest peforming algorithms as well. Having deeper trees within the ensembling process means the algorithms were able to capture more of the complex and nonlinear interactions between the features and the target variable Yu & Zhu. (2020).

However it must be stated that comparing max depth between algorithms may not always provide meaningful insights Arnould et al. (2020). Different machine learning algorithms have distinct internal mechanisms and structures. They handle complexity and overfitting differently, and their depth hyperparameters might serve various purposes. For the singular algorithm DT, max depth directly controls the depth of individual trees in the ensemble. For RF, max depth controls the depth of individual trees but is typically less sensitive to overfitting because of the ensemble's averaging effect. For LightGBM, max depth controls the depth of individual trees, but their unique algorithms also incorporate techniques like leaf-wise tree growth Zhang et al. (2019), which affects how depth is used. Hence, while deeper trees may introduce more complexity, LightGBM and RF have mechanisms in place to control this complexity effectively. Their ensemble nature and aggregation of predictions help reduce the potential bias introduced by deeper trees.

Number of Estimators

Across all subsamples and ensemble algorithms the most effective number of estimators was approximately 500. The number of estimators can be related to overfitting and diminishing returns with more estimators, the model can become overly complex and memorise the training data, leading to poor generalisation on unseen data. Overfitting can result in a high variance model that performs well on the training data but poorly on validation or test data. If the dataset contains a significant amount of noise or random variability, increasing the number of estimators can make the ensemble more sensitive to this noise. The ensemble may end up emphasising random patterns rather than meaningful signal in the data. Initially, adding more estimators can lead to substantial improvements in model accuracy. However, beyond a certain point, the improvements become marginal, and the additional computational cost may not justify the small gains in performance.

Computational Requirements Ranking 4.3

The most computationally intensive ML algorithms were RF and BR the training time for each subsample lasted between 1 – 1.5 hours and approximately 2 hours for the full data. LightGBM and CatBoost had a training time between 10 – 20 minutes which is a lot faster and efficient as compared to the other ensemble algorithms. Out of the singular algorithms LR was by far the fastest with training times being less than <1 minute. Followed by DT which was approximately <2 minutes. NN was the most time-consuming singular algorithm with training times between 2 – 10 minutes.

LightGBM and CatBoost use gradient boosting, as compared to RF and BR which use bootstrap aggregating. As seen from the results, gradient boosting is more of a time efficient algorithm as when gradient boosting constructs DTs sequentially, each new tree is trained to correct the errors made by the ensemble of previous trees. In contrast, bootstrap constructs DTs independently and in parallel. Sequential learning allows gradient boosting to adapt and allocate more resources to areas where improvement is needed, leading to faster convergence times Bentéjac et al. (2020). Additionally, LightGBM and CatBoost, have optimised approaches for handling categorical features. This can significantly reduce the computational complexity when dealing with datasets containing categorical

variables, whereas bootstrapping may require more effort for preprocessing and handling such data effectively.

What singular algorithms like DT and LR lack in performance they make up for in efficiency in computational time. They were significantly less than all other ML algorithms in terms of computational time. Simpler algorithms like these do have a practical use case, in some cases there will be computational constraints such as limitations on CPU and GPU memory. Using simpler singular ML algorithms may be a more feasible solution especially if the data contains highly linear relationships and a simpler model has a comparative performance to a more complex model Ferro et al. (2021). Additionally, simpler algorithms come with less complexity in terms of code and deployment and require less resources to maintain. In many real-time or near-real-time applications, such as online advertising, inference speed is critical. Simpler models often have faster prediction times, making them more suitable for such applications. Finally, simpler algorithms can be used for prototyping and test piloting when developing a ML solution. If researchers wanted to iterate quickly to experiment with different ideas, features, or hyperparameters. Simpler algorithms train much faster than complex ones, allowing them test hypotheses and make improvements more rapidly.

Summary of Findings 4.4

GB for feature selection was the best algorithm for selecting the most important features within the dataset. LightGBM was the best performing algorithm having the lowest MAE and the highest R^2. Referring to the original research question of whether ensemble ML algorithms are more effective than singular ML algorithms the results of the research back this statement. For feature selection the two ensemble algorithms outperformed all other singular methods. For model training the four ensemble algorithms outperformed the three singular algorithms. However, this research also investigated the computational requirements of each of these models. There was no systematic procedure to find a combined score for algorithms by consolidating their performance with the computational requirements score but there was a blatant correlation between higher performing models having a greater computational time. In consideration of computational time LightGBM still stands out as the highest performing algorithm as relatively its computational time was low in comparison to the other ensemble approaches. The value of this research is not confined to simply identifying the highest performing algorithm but should serve as a framework for readers to evaluate and weigh up the strengths and weaknesses of each approach and identify the best selection of ML algorithms based upon their situation considering aspects such as the characteristics of the data, time constraints and interpretability requirements.

Limitations 4.5

Dataset dependency 4.5.1

Dataset dependency refers to the phenomenom where the performance of ML algorithms can vary significantly based upon the specific charisterstics that comprises the dataset. As only a singular dataset was used the experimental results may have been influenced by the idiosyncrasies of the singular dataset, hence a conclusion can not be substantiated as it cannot be validated whether the observed results are due to algorithmic superiority or dataset-specific characteristics. Algorithms that happen to perform well on a particular dataset may be favoured, even if they are not fundamentally superior Flach & Savnik. (1999). This can lead to misleading conclusions about algorithm performance. Hence, the research may produce different results when replicated on different datasets. Despite this the general characteristics of the dataset can be defined (large, high-dimensional, binary data) so this may be applied to other healthcare data with similar characteristics but lacks validity as only a singular dataset was used.

Hyperparameter Tuning and Computational Capabilities 4.5.2

Hyperparameter tuning needs to be controlled for as algorithms that are extensively fine-tuned may appear superior to others simply because they have been optimised more. This will lead to a bias favouring algorithms that offer more potential for hyperparameter tuning Weerts et al. (2020). When conducting the comparisons of ML algorithms, it was discovered that the training times for BR and RF was substantially longer than other ML ensemble algorithms this meant the number of combinations of hyperparameters within the grid search was not as extensive compared to other algorithms. Hence, there is a bias favouring other algorithms as these algorithms had less of an opportunity to be optimised as a result of computational capabilities. Number of estimators on BR and RF were limited to 500 whilst for LightGBM and CatBoost a maximum of 1,000 max number of estimators were included within the grid search. This was because the training times for BR and RF were too long and having 1,000 number of estimators within the grid search significantly extended training times. However, it must be stated that no subsample in LightGBM and CatBoost required 1,000 estimators to produce optimal results, therefore it can be assumed that 1,000 estimators may be too excessive for the data as a large proportion of subsamples trained best on 500 estimators.

Additionally, there was no systematic method of defining the hyperparameters for the grid search. Instead, individual judgement was used to narrow down the ranges of search parameters after analysing the results from training various subsamples. This was because of computational limitations, as with some algorithms the training times were too long therefore it was infeasible to properly explore all likely hyperparameter combinations. If this was not a limitation a much more extensive search space would have been adopted and a sensitivity analysis would have been used. This could be done by varying a single hyperparameter while keeping others constant to better understand how a hyperparameter affects model performance.

Subsample Size and Computational Capabilities 4.5.3

When selecting features for the subsamples only three or less subsamples were created for each feature selection method. RF, GB, K-Best had three subsamples (30, 50 and 70) and over methods only had one. This was done based upon time constraints as with many subsamples the ML models training time would be too long as additional time for hyperparameter tuning must also be considered. Ideally, a greater number of subsamples would be created extending further than 70 features as from the results a greater number of features tended to perform better on ensemble algorithms. There is an element of experimental bias in including more subsamples for specific feature selection methods as this can be used to criticise why GB and RF performed better in comparison but as discussed earlier there are other underlying factors which can explain their superiority.

Data Limitations 4.5.4

As seen from the results the MAE and R^2 of the results were not ideal and would generally not be considered sufficient enough for implementation of the model. An important disclaimer must be stated in the sense that the dataset which was used was altered in a way to randomise the values of some selected features. This was done to preserve the anonymity and privacy of the users that the data was collected from. Healthcare data can be very sensitive and the data had to be encoded as a requirement for third party usage. The randomisation of features can have a huge impact on the training of ML algorithms. As randomised data by definition is intentionally designed to remove any systematic or meaningful relationships between variables, making it very difficult to identify patterns that simply do not exist. This will introduce limitations such as bias, overfitting and additional noise into training all of which are very detrimental to the peformance of ML algorithms.

Adressing Computational Capacity Limitations 4.5.5

Having a greater computational capacity can mitigate the time and resource limitations, this would allow for a greater amount of subsamples, more extensive hyperparameter tuning and a larger dataset to be trained on. This research was limited to only one device with limited GPU processing capabilities. With the implementation of cloud computing and distributed ML the computational limitations are practically limitless. Conducting future research on a distributed ML setup can reduce limitations associated with computational capacity.

Conclusion (Chapter 5)

Theoretical Contributions 5.1

The results from the research can prove useful in several different aspects. Firstly, the comparative research was performed on a real-world dataset obtained officially from the NHS's personal records. This means that this research can serve as a framework for ML applications within the healthcare industry. Additionally, it can also be applied within other domains as the characteristics of the dataset was high dimensional and sparse with lots of binary features therefore in other domains which experience datasets of these characteristics can also tailor their ML applications based upon the findings of this study.

Secondly, scalability has been a serious issue for organisations within the public sector. Therefore, improving upon resource efficiency and understanding context specific computational limitations of ML algorithms is important for organisations intending to implement ML within their operations. This research produces a cost-benefit analysis of the ML approaches flagging ML algorithms which are computationally intensive. This will prove beneficial in the decision-making process of organisations who are expecting a large volume of data and need a ML algorithm which is high performing and is also scalable. Also, it is useful for organisations operating on resource constraints, all the research conducted within the study was confined to one device with average GPU / CPU capabilities and the only software used was Python which is open source. Meaning the research can provide value to smaller organisations who may lack the resources or expertise to operate complex distributed ML networks but still seek the benefits of using ML on a smaller scale.

Thirdly, the research conducted can provide valuable insights into hyperparameter tuning for the various algorithms. Through examining the grid search hyperparameters as well as the optimal hyperparameters organisations or researchers can gain a better understanding on how to tune hyperparameter based upon the ML algorithm at hand. For instance, the optimal tree depth for LightGBM and RF was significantly greater than CatBoost this can be beneficial for researchers, who are conducting similar comparison studies, to define an initial grid search for the experimental testing.

Finally, the results of the feature selection analysis show the importance of using ensemble methods for capturing inter-feature interactions and nonlinearity in high-dimensional sparse datasets. This encourages organisations and researchers to place more of an emphasis on feature selection methods as it can drastically alter the performance of ML algorithms. Additionally, it was identified that GB was the most effective feature selection algorithm which is a valuable insight for organisations who perhaps are looking to dimensionally reduce a large dataset with similar characteristics with the one used within this research. There is also benefit for the NHS within this study as from the GB subsample the top 30/50/70 features ranked on importance can be viewed. This can be very helpful in contributing to explaining the model to stakeholders and in interpreting the model. Increasing interpretability is hugely beneficial within the healthcare sector as there is rules and regulations surrounding this aspect of ML as healthcare data is deemed to be sensitive.

Alignment with Initial Expectation 5.2

Before research was conducted the initial expectations was that ensembles would perform better than singular algorithms, this was the case in both feature selection and training algorithms. Additionally, it was predicted that the ensembles would also take much longer to train than singular algorithms. This also was proved correct based upon the results.

Also, through visualising the data types and characteristics of the dataset it was seen that there was a large proportion of binary variables. Binary data by its very nature rarely exhibits multicollinearity. In cases where the features are not multicollinear and are largely independent, regularisation may not have as much impact. Therefore, it was expected that feature selection methods such as EN and Lasso would not perform well in determining features with high importance, as regularisation is the primary use case in these methods. As predicted EN and Lasso did not perform well regarding feature selection as during the feature selection process, features with low correlation to each other are less likely to be jointly penalised, and regularisation faces a harder task to shrink feature coefficients to exactly zero.

The initial expectations were that the gradient boosted ensemble algorithms (LightGBM and CatBoost) would perform the best. This was because of the way gradient boosted algorithms use histogram-based splitting meaning they have greater capabilities to handle high-dimensional sparse data. However, the results from the research did not correctly line up with the expectations as RF outperformed the CatBoost algorithm. This may be just an outlier result or can be the product of insufficient hyperparameter tuning regarding the CatBoost algorithm.

Filling Research Gaps and Paving the Way for Future Research 5.3

This research builds upon the existing literature of comparison of ML algorithms within the healthcare domain. It substantiates much of the existing literature in the superiority of peformance of esnemble algorithms. Additionally, it fills a gap within the existing literature as the research investigates the peformance vs computational resource / time trade-off. There has been a limited amount of research in regards to this topic. Presently, ML is becoming more of an accessible tool for users and organisations that may not have technical expertise or resources to conduct large scale complex ML models. This research can benefit users who wish to use ML on a smaller scale and identify which algorithms peform best in relation to trade-offs with limitations such as scalability. Research similar to this regarding the comparison of ML algorithms can be applied to other domains such as finance or social media to investigate which algorithms / feature selection approaches work best in these domains.

Areas of Future Research 5.3.1

This research focues mainly on the comparison of singular feature selection / ML algorithms with ensemble algorithms. It would be interesting for further research to be done on the stacking of different algorithms to investigate which algorithms are complemetary to each other. Stacking has the potential to better improve predictive peformance by amplifying the strengths of stacked models whilst also mitigating the weaknesses. When stacking algorithms with diverse base models. These base models have different underlying algorithms, feature sets, or hyperparameters and can capture different aspects of the data and provide a more comprehensive view of the problem. However, this research has limitations associated with it due to the vast array of ML algorithms investigating stacking can be very computationally intensive when considering a large combination of base models. Additionally, the hyperparameter tuning will be very time consuming and will require a robust experimental setup.

Another area of future research would be to further investigate the interpretability potential of ML algorithms. In many real-world applications, ML models are used to make critical decisions, such as in healthcare, finance, and criminal justice. Users and stakeholders need to trust these models and understand the reasoning behind their predictions. Interpretability helps establish trust and makes the decision-making process more accountable. Similarly, to the research conducted within this study algorithms can be judged on several criteria, performance, scalability and interpretability. All these aspects can be ranked, and the scores can be consolidated to provide a broader more comprehensive assessment of the ML algorithms. Ideally a robust ranking system would need to be set up and algorithms would need to be trained on a variety of datasets.

Future research can be done to investigate the process of hyperparameter tuning. Hyperparameter tuning often involves exploring a large search space of possible hyperparameter values. Each hyperparameter can typically take on a range of values, and there may be many hyperparameters to optimise. The extensive combinatorial possibilities increase the time required for searches. A potential study could involve exploring hyperparameter tuning methods such as, random search, Bayesian optimisation, genetic algorithms and autoML tools. Then conduct statistical analysis to evaluate the strengths and weaknesses of each approach and overall identify the highest performing method. Conducting research within this domain can gather insights to benefit researchers and practitioners to save time / resources on their hyperparameter tuning process.

This research explored the performance and scalability on a smaller scale with GPU / CPU constraints. Future research can be done to explore the scalability of ML algorithms across sophisticated distributed ML networks with no resource constraints. The use of distributed networks opens the potential for practically limitless computational capacity leading to more sophisticated ML models trained on extremely high dimensional data. This area of research can prove particularly useful for larger organisation who deal with large scale data or are required to train models in a time constraint.

Closing Remarks 5.4

To conclude this research investigating the comparison of ML algorithms, it's clear that there is possibilities and potential for this ever-growing field of artificial intelligence. This research has been a reminder that the pursuit of the perfect algorithm does not exist. Each one has its own unique characteristics, perfectly suited to its own specific purposes and opportunities. Instead, the power lies in the hands of the practitioner who must use their expertise to mould the ML model to align with the issue at hand.

References

- Ballester, O. (2021) *An artificial intelligence definition and classification framework for Public Sector Applications*, *ACM Conferences*. Available at: https://dl.acm.org/doi/10.1145/3463677.3463709 (Accessed: 21 September 2023).
- Guo, W. *et al.* (2018) *Explaining deep learning models - a Bayesian non-parametric approach*, *arXiv.org*. Available at: https://arxiv.org/abs/1811.03422 (Accessed: 21 September 2023).
- Huang, Y. *et al.* (2021) *Application of machine learning in predicting hospital readmissions: a scoping review of the literature*, *BioMed Central*. Available at: https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-021-01284-z (Accessed: 21 September 2023).
- Khare, N. and Sait, S. (2018) *Credit Card Fraud Detection Using Machine Learning Models and Collating Machine Learning Models*. Available at: https://www.acadpubl.eu/hub/2018-118-21/articles/21b/90.pdf (Accessed: 21 September 2023).
- Lo, Y.-T. *et al.* (2022) *Predictive modeling for 14-day unplanned hospital readmission risk by using Machine Learning Algorithms - BMC Medical Informatics and decision making*, *BioMed Central*. Available at: https://bmcmedinformdecismak.biomedcentral.com/articles/10.1186/s12911-021-01639-y (Accessed: 21 September 2023).
- Patil, S., Nemade, V. and Soni, P. (2018) *Predictive Modelling for Credit Card Fraud Detection Using Data Analytics*, *Procedia Computer Science*. Available at: https://www.sciencedirect.com/science/article/pii/S1877050918309347 (Accessed: 21 September 2023).
- Pham, A. *et al.* (2020) *Predicting energy consumption in multiple buildings using machine learning for improving energy efficiency and sustainability*, *Journal of Cleaner Production*. Available at: https://www.sciencedirect.com/science/article/pii/S095965262031129X (Accessed: 21 September 2023).
- Sahin, Y. and Duman, E. (2011) *Detecting credit card fraud by ANN and logistic regression*. Available at: https://ieeexplore.ieee.org/abstract/document/5946108 (Accessed: 21 September 2023).
- Sharma, A. and Panigrahi, P. (2013) *A Review of Financial Accounting Fraud Detection based on Data Mining Techniques*, *arXiv*. Available at: https://arxiv.org/pdf/1309.3944 (Accessed: 21 September 2023).
- Zeyu, W. *et al.* (2018) *Random Forest based hourly building energy prediction*, *Energy and Buildings*. Available at: https://www.sciencedirect.com/science/article/pii/S0378778818311290 (Accessed: 21 September 2023).
- Bekkerman, R. *et al.* (2012) *Scaling Up Machine Learning*, *Cambridge University Press*. Available at: https://assets.cambridge.org/97805211/92248/frontmatter/9780521192248_frontmatter.pdf (Accessed: 21 September 2023).
- Cruz, A. *et al.* (2021) *Promoting Fairness through Hyperparameter Optimization*, *IEEE*. Available at: https://ieeexplore.ieee.org/document/9679036 (Accessed: 21 September 2023).

- Jesus, S. *et al.* (2022) *Turning the Tables: Biased, Imbalanced, Dynamic Tabular Datasets for ML Evaluation*. Available at: https://proceedings.neurips.cc/paper_files/paper/2022/file/d9696563856bd350e4e7ac5e5812f23c-Paper-Datasets_and_Benchmarks.pdf (Accessed: 21 September 2023).
- Jiang, S. *et al.* (2018) *An integrated machine learning framework for hospital readmission prediction*, *Knowledge-Based Systems*. Available at: https://www.sciencedirect.com/science/article/pii/S0950705118300443#sec0028 (Accessed: 21 September 2023).
- Liu, B. (2013) *Scalable sentiment classification for Big Data analysis using Naïve Bayes Classifier*, *IEEE*. Available at: https://ieeexplore.ieee.org/abstract/document/6691740/footnotes#footnotes (Accessed: 21 September 2023).
- Min, X. *et al.* (2019) *Predictive modeling of the hospital readmission risk from patients' claims data using Machine Learning: A case study on COPD*, *Nature News*. Available at: https://www.nature.com/articles/s41598-019-39071-y (Accessed: 21 September 2023).
- Mohammad, R. *et al.* (2018) *Machine Learning Techniques for Highly Imbalanced Credit Card Fraud Detection: A Comparative Study*, *Springer Link*. Available at: https://link.springer.com/chapter/10.1007/978-3-319-97310-4_27 (Accessed: 21 September 2023).
- Nikou, M. *et al.* (2019) *Stock price prediction using DEEP learning algorithm and its comparison with machine learning algorithms*, *Wiley*. Available at: https://onlinelibrary.wiley.com/doi/10.1002/isaf.1459 (Accessed: 21 September 2023).
- Radzi, S. *et al.* (2021) *Hyperparameter Tuning and Pipeline Optimization via Grid Search Method and Tree-Based AutoML in Breast Cancer Prediction*, *JPM*. Available at: https://www.mdpi.com/2075-4426/11/10/978 (Accessed: 21 September 2023).
- Wolpert, D. and Macready, W. (1997) *No free lunch theorems for optimization*. Available at: https://ieeexplore.ieee.org/abstract/document/585893 (Accessed: 21 September 2023).
- Adadi, A. and Berrada, M. (2018) *Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)*, *IEEE*. Available at: https://ieeexplore.ieee.org/document/8466590 (Accessed: 21 September 2023).
- Banet, T.A. and Nafria, E. (2015) *Stability and scalability in decision trees*, *Springer Link*. Available at: https://link.springer.com/article/10.1007/BF03354613 (Accessed: 21 September 2023).
- Elkan, C. (1997) *Naive Bayesian Learning*, *Harvard University*. Available at: https://home.engineering.iastate.edu/julied/classes/ee547/Handouts/naive-bayesian-learning.pdf (Accessed: 21 September 2023).
- Elshawi, R. *et al.* (2019) *On the interpretability of machine learning-based model for predicting hypertension*, *Springer Link*. Available at: https://link.springer.com/article/10.1186/s12911-019-0874-0 (Accessed: 21 September 2023).
- Karatza, P. *et al.* (2021) *Interpretability methods of machine learning algorithms with applications in breast cancer diagnosis*, *IEEE*. Available at: https://ieeexplore.ieee.org/document/9630556 (Accessed: 21 September 2023).

- Katuwal, G. and Chen, R. (2016) *Machine Learning Model Interpretability for Precision Medicine*, *arXiv*. Available at: https://arxiv.org/abs/1610.09045 (Accessed: 21 September 2023).
- Pavlov, D. *et al.* (2002) *Scaling-up support vector machines using boosting algorithm*, *IEEE*. Available at: https://ieeexplore.ieee.org/abstract/document/906052 (Accessed: 21 September 2023).
- Razzaghi, T. and Safro, I. (2015) *Scalable Multilevel Support Vector Machines*, *ScienceDirect*. Available at: https://www.sciencedirect.com/science/article/pii/S1877050915011898 (Accessed: 21 September 2023).
- Wu, H. *et al.* (2021) *Interpretable Machine Learning for COVID-19: An Empirical Study on Severity Prediction Task*, *IEEE*. Available at: https://ieeexplore.ieee.org/document/9465748 (Accessed: 21 September 2023).
- Wu, Y. *et al.* (2023) *Interpretable Machine Learning for Personalized Medical Recommendations: A LIME-Based Approach*, *MDPI*. Available at: https://www.mdpi.com/2075-4418/13/16/2681 (Accessed: 21 September 2023).
- Bellman, R. (1957) *Dynamic Programming, Science*. Available at: https://www.science.org/doi/10.1126/science.153.3731.34 (Accessed: 22 September 2023).
- Borkin, D. *et al.* (2019) *Impact of data normalization on classification model accuracy*, *Sciendo*. Available at: https://sciendo.com/article/10.2478/rput-2019-0029 (Accessed: 22 September 2023).
- Gnat, S. (2021) *Impact of categorical variables encoding on property mass valuation*, *Procedia Computer Science*. Available at: https://www.sciencedirect.com/science/article/pii/S1877050921018664 (Accessed: 22 September 2023).
- Masconi, K.L. *et al.* (2015) *Effects of different missing data imputation techniques on the performance of undiagnosed diabetes risk prediction models in a mixed-ancestry population of South Africa*, *PLOS ONE*. Available at: https://journals.plos.org/plosone/article?id=10.1371%2Fjournal.pone.0139210 (Accessed: 22 September 2023).
- Nyitrai, T. and Virág, M. (2018) *The effects of handling outliers on the performance of bankruptcy prediction models*, *Socio-Economic Planning Sciences*. Available at: https://www.sciencedirect.com/science/article/pii/S003801211730232X (Accessed: 22 September 2023).
- Zhou, S. *et al.* (2017) *Machine learning on Big Data: Opportunities and challenges*, *Neurocomputing*. Available at: https://www.sciencedirect.com/science/article/pii/S0925231217300577 (Accessed: 22 September 2023).
- *Deep Learning, Neural Networks* (2018) *Springer Link*. Available at: https://link.springer.com/chapter/10.1007/978-3-319-72347-1_23 (Accessed: 22 September 2023).
- Hancock, J.T. and Koshgoftaar, T.M. (2020) *CatBoost for big data: an interdisciplinary review*, *Springer Open*. Available at: https://journalofbigdata.springeropen.com/articles/10.1186/s40537-020-00369-8 (Accessed: 22 September 2023).

- Ke, G. (2017) *LightGBM: A Highly Efficient Gradient Boosting Decision Tree*, *Semantic Scholar*. Available at: https://www.semanticscholar.org/paper/LightGBM%3A-A-Highly-Efficient-Gradient-Boosting-Tree-Ke-Meng/497e4b08279d69513e4d2313a7fd9a55dfb73273 (Accessed: 22 September 2023).
- Khiari, J. *et al.* (2018) *MetaBags: Bagged Meta-Decision Trees for Regression*, *Springer Link*. Available at: https://link.springer.com/chapter/10.1007/978-3-030-10925-7_39 (Accessed: 22 September 2023).
- Murphy, A. and Moore, F. (2019) *Random Forest (machine learning)*, *Radiopaedia*. Available at: https://radiopaedia.org/articles/random-forest-machine-learning?lang=gb (Accessed: 22 September 2023).
- Sazonau, V. (2012) *Implementation and Evaluation of a Random Forest Machine Learning Algorithm*, *Semantic Scholar*. Available at: https://www.semanticscholar.org/paper/Implementation-and-Evaluation-of-a-Random-Forest-Sazonau/71bb505bfee1b33a04a513e1320482ca3538ab75 (Accessed: 22 September 2023).
- Torstensson, E. (2017) *Using LASSO regularization as a feature selection tool.*, *Semantic Scholar*. Available at: https://www.semanticscholar.org/paper/Using-LASSO-regularization-as-a-feature-selection-Torstensson/bedeb4fff055dc430620f419dd00fed5fcd19042 (Accessed: 22 September 2023).
- Xu, Z. *et al.* (2014) *Gradient boosted feature selection: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and data mining*, *ACM Conferences*. Available at: https://dl.acm.org/doi/10.1145/2623330.2623635 (Accessed: 22 September 2023).
- Yan, J. *et al.* (2021) *LightGBM: accelerated genomically designed crop breeding through ensemble learning*, *BMC*. Available at: https://genomebiology.biomedcentral.com/articles/10.1186/s13059-021-02492-y (Accessed: 22 September 2023).
- Zou, H. and Hastie, T. (2005) *Regularization and Variable Selection Via the Elastic Net*, *Oxford Academic*. Available at: https://academic.oup.com/jrsssb/article/67/2/301/7109482 (Accessed: 22 September 2023).
- Arnould, L. (2021) *Analyzing the tree-layer structure of Deep Forests*, *arXiv*. Available at: https://arxiv.org/abs/2010.15690 (Accessed: 22 September 2023).
- Bentejac, C. (2020) *A comparative analysis of gradient boosting algorithms*, *Springer Link*. Available at: https://link.springer.com/article/10.1007/s10462-020-09896-5 (Accessed: 22 September 2023).
- Canedo, V.B. and Betanzos, A.A. (2019) *Ensembles for feature selection: A review and future trends*, *Science Direct*. Available at: https://www.sciencedirect.com/science/article/pii/S1566253518303440?via%3Dihub#sec0011 (Accessed: 22 September 2023).
- Canedo, V.B. *et al.* (2016) *Feature selection for high-dimensional data*, *Springer Link*. Available at: https://link.springer.com/article/10.1007/s13748-015-0080-y (Accessed: 22 September 2023).
- Elgimati, Y. (2020) *Weighted Bagging in Decision Trees: Data Mining*, *Semantic Scholar*. Available at: https://www.semanticscholar.org/paper/Weighted-Bagging-in-Decision-Trees%3A-Data-Mining-Elgimati/9945567c3ac5a3ff9595fa9b485393957d47e076 (Accessed: 22 September 2023).

- Ferro, M. (2021) *Towards a sustainable artificial intelligence: A case study of energy efficiency in decision tree algorithms*, *Wiley*. Available at: https://onlinelibrary.wiley.com/doi/full/10.1002/cpe.6815 (Accessed: 22 September 2023).
- Flach, P.A. and Savnik, I. (1999) *Database Dependency Discovery: A Machine Learning Approach*, *AI Commun*. Available at: https://www.semanticscholar.org/paper/Database-Dependency-Discovery%3A-A-Machine-Learning-Flach-Savnik/c2f4c6d7e06da14c4b3ce3a9b97394a64708dc52 (Accessed: 22 September 2023).
- Weerts, H.J. *et al.* (2020) *Importance of Tuning Hyperparameters of Machine Learning Algorithms*, *ArXiv*. Available at: https://www.semanticscholar.org/paper/Importance-of-Tuning-Hyperparameters-of-Machine-Weerts-Mueller/a942b64b4b43f15ae5b12e3272cd588a26d2acd5 (Accessed: 22 September 2023).
- Yan, W. *et al.* (2009) *Feature selection based on bagging ensemble learning algorithm*, *IET*. Available at: https://digital-library.theiet.org/content/conferences/10.1049/cp.2009.2058 (Accessed: 22 September 2023).
- Yu, T. and Zhu, H. (2020) *Hyper-Parameter Optimization: A Review of Algorithms and Applications*, *ArXiv*. Available at: https://www.semanticscholar.org/paper/Hyper-Parameter-Optimization%3A-A-Review-of-and-Yu-Zhu/a48fd38cc34f8ffe9bcf043eafe11289627dd91a (Accessed: 22 September 2023).
- Zhang, J. *et al.* (2019) *LightGBM: An Effective and Scalable Algorithm for Prediction of Chemical Toxicity–Application to the Tox21 and Mutagenicity Data Sets*, *ACSPublications*. Available at: https://pubs.acs.org/doi/10.1021/acs.jcim.9b00633 (Accessed: 22 September 2023).
- Tae, C.M. and Hung, P.D. (2019) *Comparing ML algorithms on financial fraud detection: Proceedings of the 2019 2nd International Conference on Data Science and Information Technology*, *ACM Other conferences*. Available at: https://dl.acm.org/doi/10.1145/3352411.3352416 (Accessed: 25 September 2023).
- Salam, A. and Hiaboui, A.E. (2018) *Comparison of Machine Learning Algorithms for the Power Consumption Prediction: - Case Study of Tetouan city*, *IEEE*. Available at: https://www.semanticscholar.org/paper/Comparison-of-Machine-Learning-Algorithms-for-the-%3A-Salam-Hibaoui/177512e766fe5d8624a1b3e93abd11082ac37b3f (Accessed: 25 September 2023).
- Saab, A. *et al.* (2020) *Comparison of Machine Learning Algorithms for Classifying Adverse-Event Related 30-Day Hospital Readmissions: Potential Implications for Patient Safety*, *IOS*. Available at: https://ebooks.iospress.nl/publication/54591 (Accessed: 25 September 2023).
- Singh, K. *et al.* (2014) *Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests*, *Science Direct*. Available at: https://www.sciencedirect.com/science/article/pii/S0020025514003570?via%3Dihub#s0035 (Accessed: 25 September 2023).
- Yeung, K. and Lodge, M. (2019) *Administration by Algorithm? Public Management Meets Public Sector Machine Learning*, *SSRN*. Available at: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3375391 (Accessed: 26 September 2023).
- Veale, M. *et al.* (2018) *Fairness and Accountability Design Needs for Algorithmic Support in High-Stakes Public Sector Decision-Making*, *ACM*. Available at: https://dl.acm.org/doi/10.1145/3173574.3174014 (Accessed: 26 September 2023).

Appendix

Data Preparation and Cleaning (Figure 1.1 – 6.1)

Merging (Figure 1.1)

```
In [2]: df1 = pd.read_csv("GnA_sample_0.csv")
        df2 = pd.read_csv("GnA_sample_1.csv")
        df3 = pd.read_csv("GnA_sample_2.csv")
        df4 = pd.read_csv("GnA_sample_3.csv")
        df5 = pd.read_csv("GnA_sample_4.csv")

In [3]: result = pd.concat([df1, df2, df3, df4, df5], ignore_index=True)
```

Figure 1.1

Missing Values (Figure 2.1 – 2.7)

```
In [6]: total_missing_values = df.isnull().sum().sum()
        print("\nTotal number of missing values in the entire DataFrame:", total_missing_values)

        Total number of missing values in the entire DataFrame: 172504
```

Figure 2.1

```
In [7]: missing_values_count = df.isnull().sum()

        # Filter columns with more than 1000 missing values
        columns_with_over_1000_missing = missing_values_count[missing_values_count > 1000]

        print("Columns with over 1000 missing values:")
        print(columns_with_over_1000_missing)

        Columns with over 1000 missing values:
        eligibleGP       49995
        rgp              49995
        selGP            49995
        ind_ethnicity    19547
        dtype: int64
```

Figure 2.2

```
In [8]: columns_to_remove = ['eligibleGP', 'rgp', 'selGP', 'ind_ethnicity']
        df = df.drop(columns=columns_to_remove)
```

Figure 2.3

```
In [9]: df.dropna(inplace=True)
```

Figure 2.4

```
In [12]: # Calculate the sum of total missing values in the DataFrame
         total_missing = df.isnull().sum().sum()

         # Print the sum of total missing values
         print("Total missing values:", total_missing)

         Total missing values: 0
```

Figure 2.5

```
In [3]: specific_value = 0

        # Calculate percentage of values equal to '0'
        percentage = (df.applymap(lambda x: x == specific_value).sum().sum() / df.size) * 100

        print(f"Percentage of values equal to {specific_value}: {percentage:.2f}%")

        Percentage of values equal to 0: 87.05%
```

Figure 2.6

```
In [4]: columns_with_two_unique_values = []

        # Iterate through all columns
        for column in df.columns:
            if df[column].nunique() == 2:
                columns_with_two_unique_values.append(column)

        print("Columns with only two unique values:", columns_with_two_unique_values)
```

```
_A79', 'A80_A89', 'A90_A99', 'B00_B09', 'B15_B19', 'B25_B34', 'B35_B49', 'B50_B64', 'B65_B83', 'B85_B99', 'C00_C14', 'C15_C2
6', 'C30_C39', 'C43_C44', 'C45_C49', 'C50', 'C51_C58', 'C60_C63', 'C64_C68', 'C69_C72', 'C73_C80', 'C81_C96', 'D00_D48', 'D50
_D64', 'D65_D89', 'E00_E07', 'E10_E14', 'E15_E90', 'F00_F03', 'F04_F09', 'F10_F19', 'F20_F29', 'F30_F39', 'F40_F69', 'F70_F7
9', 'F80_F99', 'G00_G09', 'G10_G14', 'G20_G26', 'G35_G37', 'G40_G47', 'G50_G73', 'G80_G83', 'H00_H06', 'H10_H13', 'H25_H28',
'H40_H42', 'H60_H95', 'I00_I09', 'I10_I15', 'I20_I25', 'I26_I28', 'I30_I52', 'I60_I69', 'I70_I79', 'I80_I89', 'I95_I99', 'J00
_J06', 'J09_J18', 'J20_J22', 'J30_J39', 'J40_J47', 'J60_J70', 'J80_J99', 'K00_K14', 'K20_K31', 'K35_K38', 'K40_K46', 'K50_K5
2', 'K55_K64', 'K65_K67', 'K70_K77', 'K80_K87', 'K90_K93', 'L00_L14', 'L20_L30', 'L40_L45', 'L50_L54', 'M00_M25', 'M30_M36',
'M40_M54', 'M60_M79', 'M80_M94', 'M95_M99', 'N00_N08', 'N17_N19', 'N20_N23', 'N25_N29', 'N30_N39', 'N40_N51', 'N60_N64', 'N70
_N77', 'N80_N98', 'N99', 'O00_O08', 'O10_O75', 'O80_O84', 'P00_P04', 'P05_P96', 'Q00_Q89', 'Q90_Q99', 'R00_R09', 'R10_R19',
'R20_R23', 'R25_R29', 'R30_R39', 'R40_R46', 'R47_R49', 'R50_R68', 'R69', 'R70_R89', 'R90_R94', 'S00_S09', 'S10_S19', 'S20_S2
9', 'S30_S39', 'S40_S49', 'S50_S59', 'S60_S69', 'S70_S79', 'S80_S89', 'S90_S99', 'T00_T07', 'T08_T14', 'T15_T19', 'T20_T32',
'T36_T50', 'T51_T65', 'T66_T78', 'T79', 'T80_T88', 'T90_T98', 'VVV', 'WWW', 'XXX', 'YYY', 'Z00_Z13', 'Z20_Z29', 'Z30_Z39', 'Z
40_Z54', 'Z55_Z65', 'Z70_Z76', 'Z80_Z99', 'U_Unclassified', 'A00_B99', 'C00_D48', 'D50_D89', 'E00_E90', 'F00_F99', 'G00_G99',
'H00_H59', 'H60_H95_CH', 'I00_I99', 'J00_J99', 'K00_K93', 'L00_L99', 'M00_M99', 'N00_N99', 'O00_O99', 'P00_P96', 'Q00_Q99',
'R00_R99', 'S00_T98', 'V01_Y98', 'Z00_Z99', 'C00_D48_x_A00_B99', 'D50_D89_x_A00_B99', 'D50_D89_x_C00_D48', 'E00_E90_x_A00_B9
9', 'E00_E90_x_C00_D48', 'E00_E90_x_D50_D89', 'F00_F99_x_A00_B99', 'F00_F99_x_C00_D48', 'F00_F99_x_D50_D89', 'F00_F99_x_E00_E
90', 'G00_G99_x_A00_B99', 'G00_G99_x_C00_D48', 'G00_G99_x_D50_D89', 'G00_G99_x_E00_E90', 'G00_G99_x_F00_F99', 'H00_H59_x_A00_
B99', 'H00_H59_x_C00_D48', 'H00_H59_x_D50_D89', 'H00_H59_x_E00_E90', 'H00_H59_x_F00_F99', 'H00_H59_x_G00_G99', 'H60_H95_CH_x_
A00_B99', 'H60_H95_CH_x_C00_D48', 'H60_H95_CH_x_D50_D89', 'H60_H95_CH_x_E00_E90', 'H60_H95_CH_x_F00_F99', 'H60_H95_CH_x_G00_G
99', 'H60_H95_CH_x_H00_H59', 'I00_I99_x_A00_B99', 'I00_I99_x_C00_D48', 'I00_I99_x_D50_D89', 'I00_I99_x_E00_E90', 'I00_I99_x_F
```

```
In [5]: count = len(columns_with_two_unique_values)

        print("Number of binary columns:", count)
```

```
Number of binary columns: 1028
```

Figure 2.7

Outlier Removal (Figure 3.1 – 3.6)

```
In [17]: # Create a histogram for the variable 'total_cost'
         plt.hist(df['total_cost'], bins=15, edgecolor='black')
         plt.xlabel('total_cost')
         plt.ylabel('Frequency')
         plt.title('Histogram of total_cost')
         plt.show()
```

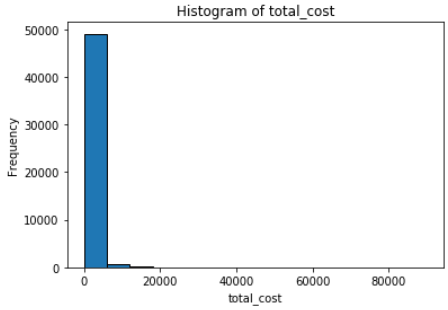


Figure 3.1

```
In [18]: target_column = 'total_cost'

         # Count the number of values equal to '0' in the 'total_cost' column
         count_zeros = (df[target_column] == 0).sum()

         # Print the count of values equal to '0'
         print("Number of values equal to '0' in", target_column, ":", count_zeros)
```

```
Number of values equal to '0' in total_cost : 28902
```

Figure 3.2

In [29]:
```
# Specify the variable you want to create a box plot for
target_variable = 'total_cost'

# Create a box plot
plt.boxplot(df[target_variable])
plt.title('Box Plot of ' + target_variable)
plt.ylabel(target_variable)
plt.show()
```

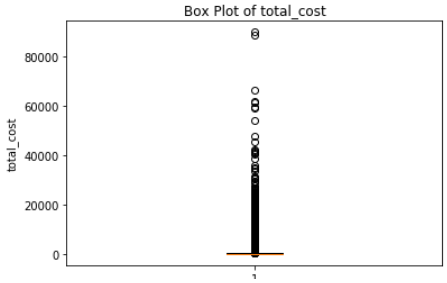


Figure 3.3

In [48]:
```
# Specify the column for which you want to calculate the IQR
target_column = 'total_cost'

# Calculate the first quartile (25th percentile)
q1 = df[target_column].quantile(0.25)

# Calculate the third quartile (75th percentile)
q3 = df[target_column].quantile(0.75)

# Calculate the interquartile range (IQR)
iqr = q3 - q1

# Print the calculated IQR
print("Interquartile Range (IQR) for", target_column, ":", iqr)
print("25th percentile", target_column, ":", q1)
print("75th percentile", target_column, ":", q3)
```

```
Interquartile Range (IQR) for total_cost : 246.0
25th percentile total_cost : 0.0
75th percentile total_cost : 246.0
```

Figure 3.4

In [49]:
```
# Specify the column you want to check and the threshold value
target_column = 'total_cost'
threshold = 615

# Count the number of values greater than the threshold in the specified column
count_greater_than_threshold = (df[target_column] > threshold).sum()

# Print the count of values greater than the threshold
print("Number of values greater than", threshold, "in", target_column, ":", count_greater_than_threshold)
```

```
Number of values greater than 615 in total_cost : 7592
```

Figure 3.5

In [51]:
```
# Specify target column
target_column = 'total_cost'
threshold = 615

# Remove rows where the values in the specified column are greater than the threshold
df_filtered = df[df[target_column] <= threshold]

df = df_filtered
```

Figure 3.6

Normalising Variables (Figure 4.1)

In [14]:
```
# Specify the columns to exclude from normalization
columns_to_exclude = ['HH_type', 'hhtype_merge', 'GP_merge', 'GPqof_merge', 'Morb_merge', 'Morbcount_merge', 'ethni_merge', 'LSOA

# Create a DataFrame to normalize (excluding specified columns)
df_to_normalize = df.drop(columns=columns_to_exclude)

# Normalize the columns using the formula (x - min) / (max - min)
for column in df_to_normalize.columns:
    min_val = df_to_normalize[column].min()
    max_val = df_to_normalize[column].max()
    df[column] = (df[column] - min_val) / (max_val - min_val)

# Print the normalized DataFrame
print(df)
```

Figure 4.1

Converting Categorical Values to Dummies (Figure 5.1)

In [1]:
```
# Perform one-hot encoding
df_encoded = pd.get_dummies(df, columns=['age_cat'], prefix=['age_cat'])

df = df_encoded

print(df)
```

Figure 5.1

Removing Constant Features (Figure 6.1)

In [ ]:
```
# Identify and remove constant columns
constant_columns = [col for col in df.columns if df[col].nunique() == 1]
df.drop(columns=constant_columns, inplace=True)

print(df)
```

Figure 6.1

Feature Selection (Figure 7.1 – 11.4)

Gradient Boosting (Figure 7.1 - 7.5)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np

df = pd.read_csv("Norm_All_Float_NoMissing_NoTC_NoCF.csv")

X = df.drop(columns=['total_cost'])
y = df['total_cost']
```

In [2]:
```
from sklearn.ensemble import GradientBoostingRegressor
```

In [3]:
```
# Create a Gradient Boosting Regressor model
gb_regressor = GradientBoostingRegressor(n_estimators=100, random_state=42)

# Train the model
gb_regressor.fit(X, y)
```

Out[3]:
```
GradientBoostingRegressor(random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [4]:
```
# Get feature importances
importances = gb_regressor.feature_importances_
```

In [5]:
```
# Sort importances in descending order
sorted_indices = np.argsort(importances)[::-1]
```

Figure 7.1

In [6]:
```
# Define the number of top features to keep
top_n_features = 30

# Get the indices of the top N important features
top_feature_indices = sorted_indices[:top_n_features]

# Print the selected top features
selected_features = X.columns[top_feature_indices]
print("Top Selected Features:")
for feature in selected_features:
    print(feature)
```

Figure 7.2

In [7]:
```
# Define the number of top features to keep
top_n_features = 50  # Change this to 30 or any other desired number

# Get the indices of the top N important features
top_feature_indices = sorted_indices[:top_n_features]

# Print the selected top features
selected_features = X.columns[top_feature_indices]
print("Top Selected Features:")
for feature in selected_features:
    print(feature)
```

Figure 7.3

In [8]:
```
# Define the number of top features to keep
top_n_features = 70  # Change this to 30 or any other desired number

# Get the indices of the top N important features
top_feature_indices = sorted_indices[:top_n_features]

# Print the selected top features
selected_features = X.columns[top_feature_indices]
print("Top Selected Features:")
for feature in selected_features:
    print(feature)
```

Figure 7.4

WhiteIrish, countdiag, WhiteOtherWhite, countdiag_cat, asp10, asp7, asp11, asp9, WhiteandBlackCaribbean, asp4, asp8, WhiteandBlackAfrican, WhiteandAsian, male, asp0, Z00_Z99_x_O00_O99_6d, asp6, Z80_Z99, asp3, asp1, Whiteother, asp5, F00_F03, E10_E14, C00_D48, countdiag1, G00_G99_6d, IoDRoaddistancetoaGPsurgeryin, QoFPrevalence1819_RA, M00_M99_6d, ProportionSeparatedbutstill, ProportionSinglenevermarried, QoFPrevalence1819_OB, case_id, H00_H59_6d, BB_5009_Pseudo_IDN, K20_K31_6d, K90_K93_c_D2, Longtermunemployed, countdiag6d1, countdiag8, QoFPrevalence1819_HYP, DWPalternativeclaimant, traveldist, QoFPrevalence1819_CHD, Mixedethnicgroup, gdirectdists, IoDChildrenandYoungPeopleSubdo, M30_M36, G00_G99, K20_K31, I26_I28_6d, Logpopvar, IoDHousingaffordabilityindicator, M40_M54_6d, GPiMenB_12m, asp2, IoDEducationSkillsandTrainingS, QoFPrevalence1819_DM, EconomicallyInactiveLongTerm, S00_T98_x_M00_M99, Proportionofstudentsinpopula, QoFPrevalence1819_LVSD, J00_J99_x_H00_H59, AsianorAsianBritish, M30_M36_6d, IoDRoaddistancetoaprimaryschoo, GPsWhenlastgeneralpracticeappoi, Proportionaged1674neverwork, WWW

Figure 7.5

Random Forest (Figure 8.1 – 8.5)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np

df = pd.read_csv("Norm_All_Float_NoMissing_NoTC_NoCF.csv")

X = df.drop(columns=['total_cost'])  # 'target_column' should be replaced with the actual target column name
y = df['total_cost']
```

In [2]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [3]:
```
# Create a Random Forest Regressor model
rf_regressor = RandomForestRegressor(n_estimators=100, random_state=42)

# Train the model
rf_regressor.fit(X, y)
```

Out[3]:
```
RandomForestRegressor(random_state=42)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [6]:
```
# Get feature importances
importances = rf_regressor.feature_importances_
sorted_indices = np.argsort(importances)[::-1]
```

Figure 8.1

In [7]:
```
# Define the number of top features to keep
top_n_features = 30  # Adjust as needed

# Get the indices of the top N important features
top_feature_indices = sorted_indices[:top_n_features]

# Print the selected features
selected_features = X.columns[top_feature_indices]
print("Selected Features:")
for feature in selected_features:
    print(feature)
```

Figure 8.2

In [9]:
```
# Define the number of top features to keep
top_n_features = 50  # Adjust as needed

# Get the indices of the top N important features
top_feature_indices = sorted_indices[:top_n_features]

# Print the selected features
selected_features = X.columns[top_feature_indices]
print("Selected Features:")
for feature in selected_features:
    print(feature)
```

Figure 8.3

In [10]:
```
# Define the number of top features to keep
top_n_features = 70  # Adjust as needed

# Get the indices of the top N important features
top_feature_indices = sorted_indices[:top_n_features]

# Print the selected features
selected_features = X.columns[top_feature_indices]
print("Selected Features:")
for feature in selected_features:
    print(feature)
```

Figure 8.4

WhiteIrish, BB_5009_Pseudo_IDN, case_id, r, countdiag, IoDRoadtrafficaccidentsindicator, IoDStayingonineducationpost16, IoDHousinginpoorconditionindica, IoDRoaddistancetoaprimaryschoo, IoDRoaddistancetoapostofficei, Proportionaged1674inroutine, Logpopvar, IoDRoaddistancetoaGPsurgeryin, Proportionaged1674insemiro, ProportionSeparatedbutstill, IoDBarrierstoHousingandServices, Mixedethnicgroup, ProportionSinglePensionerHous, ProportionWidowed, Whiteother, ProportionDivorced, IoDRoaddistancetogeneralstoreo, LSOA_, IoDHousingaffordabilityindicator, countdiag_cat, IoDYearsofpotentiallifelostind, IoDEntrytohighereducationindica, Longtermunemployed, GPsWhenlastgeneralpracticeappoi, IoDChildrenandYoungPeopleSubdo, IoDHouseswithoutcentralheatingi, Proportionaged1674neverwork, QoFPrevalence1819_CVDPP, traveldist, Proportionofstudentsinpopula, IoDAcutemorbidityindicator, gtraveldurs, gHSfcriticalcarebedsoccupieds, IoDGeographicalBarriersSubdomain, QoFPrevalence1819_OST, asp0, GPsLongtermphysicalormentalhea, ProportionSinglenevermarried, QoFPrevalence1819_MH, directdist, IoDMoodandanxietydisordersindic, GPscarer2018, traveldur, BlackorBlackBritish, QoFPrevalence1819_LVSD, AsianorAsianBritish, GPiMMR2_5y, QoFPrevalence1819_DEP, QoFPrevalence1819_RA, DWPalternativeclaimant, IoDIncomeDeprivationAffectingChi, asp1, GPwregistrationsperFTEexcluding, IoDHouseholdovercrowdingindicato, gHSfOfwhichdedicateddaycases, Proportionunstandardisedwit, countGP, asp2, Otherethnicgroup, GPsLongtermhealthconditionS, gdirectdists, QoFPrevalence1819_DEM, IoDAirqualityindicator, gtraveldists, QoFPrevalence1819 AST

Figure 8.5

Lasso Regularisation (Figure 9.1 – 9.2)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np
```

In [2]:
```
df = pd.read_csv("Norm_All_Float_NoMissing_NoTC_NoCF.csv")
```

In [3]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Lasso
```

In [4]:
```
X = df.drop(columns=['total_cost'])
y = df['total_cost']
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [6]:
```
lasso = Lasso(alpha=1.5)  # You can adjust the regularization strength (alpha) as needed
lasso.fit(X_train_scaled, y_train)
```

Out[6]:
```
Lasso(alpha=1.5)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [7]:
```
# Get the coefficients of the Lasso model
coefficients = lasso.coef_

# Find the indices of non-zero coefficients (selected features)
selected_feature_indices = np.where(coefficients != 0)[0]
```

Figure 9.1

In [8]:
```
# Get the column names of the selected features
selected_feature_names = X.columns[selected_feature_indices]

# Print the selected feature names
print("Selected feature names:", selected_feature_names)
```

```
Selected feature names: Index(['male', 'QoFPrevalence1819_RA', 'QoFPrevalence1819_AST',
       'gHSfcriticalcarebedsoccupieds', 'DWPalternativeclaimant',
       'Proportionaged1674inroutine', 'Proportionunstandardisedwit',
       'Longtermunemployed', 'Proportionofstudentsinpopula', 'A15_A19',
       ...
       'R20_R23_c_D2_6d', 'R40_R46_c_D2_6d', 'S00_S09_c_D2_6d',
       'S50_S59_c_D2_6d', 'WWW_c_D2_6d', 'countdiag6d1', 'AGE_a0',
       'AGE_a70_79', 'HH_type_Multi-adult-child', 'HH_type_Other communal'],
      dtype='object', length=114)
```

Figure 9.2

Elastic Net (Figure 10.1 – 10.2)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np

df = pd.read_csv("Norm_All_Float_NoMissing_NoTC_NoCF.csv")

X = df.drop(columns=['total_cost'])  # 'target_column' should be replaced with the actual target column name
y = df['total_cost']
```

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import ElasticNet
```

In [3]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [4]:
```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [5]:
```
elastic_net = ElasticNet(alpha=5, l1_ratio=0.5)
elastic_net.fit(X_train_scaled, y_train)
```

Out[5]:
```
ElasticNet(alpha=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [6]:
```
selected_features_indices = elastic_net.coef_ != 0
selected_features = X.columns[selected_features_indices]
```

Figure 10.1

In [7]:
```
# Get the coefficients of the Elastic Net model
coefficients = elastic_net.coef_

# Find the indices of non-zero coefficients (selected features)
selected_feature_indices = np.where(coefficients != 0)[0]

# Get the column names of the selected features
selected_feature_names = X.columns[selected_feature_indices]

# Print the selected feature names
print("Selected feature names:", selected_feature_names)
```

```
Selected feature names: Index(['male', 'QoFPrevalence1819_HYP', 'QoFPrevalence1819_CHD',
       'QoFPrevalence1819_HF', 'QoFPrevalence1819_STIA',
       'QoFPrevalence1819_RA', 'QoFPrevalence1819_AST', 'DLAPIP',
       'Proportionaged1674inroutine', 'Proportionunstandardisedwit',
       ...
       'countdiag6d5', 'countdiag6d6', 'countdiag6d7', 'countdiag6d8',
       'AGE_a0', 'AGE_a65_69', 'AGE_a70_79', 'AGE_a80_84',
       'HH_type_Other communal', 'HH_type_Two adults diff gender'],
      dtype='object', length=149)
```

Figure 10.2

K-Best (Figure 11.1 – 11.4)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np
```

In [2]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif
```

In [3]:
```
df = pd.read_csv("Norm_All_Float_NoMissing_NoTC_NoCF.csv")
```

In [4]:
```
X = df.drop(columns=["total_cost"])
Y = df["total_cost"]
```

In [5]:
```
k_best_selector = SelectKBest(score_func=f_classif, k=30)

# Fit the selector to your data
k_best_selector.fit(X, Y)
```

Figure 11.1

In [6]:
```
# Transform the feature matrix to keep only the selected k best features
X_selected = k_best_selector.transform(X)

# Get a boolean mask of selected features
selected_mask = k_best_selector.get_support()

# Indexes of the selected features
selected_feature_indices = [i for i, selected in enumerate(selected_mask) if selected]

# Create a new DataFrame with the selected features
X_selected_df = X.iloc[:, selected_feature_indices]

# Print the column names of the selected features
selected_feature_columns = X_selected_df.columns
print("Selected feature columns:", selected_feature_columns)
```

```
Selected feature columns: Index(['C50', 'C64_C68', 'Q00_Q99_x_H00_H59', 'V01_Y98_x_P00_P96', 'C50_c_D2',
       'C64_C68_c_D2', 'H60_H95_c_D2', 'K90_K93_c_D2', 'M40_M54_c_D2',
       'N70_N77_c_D2', 'S00_S09_c_D2', 'S80_S89_c_D2', 'Z00_Z13_c_D2',
       'U_Unclassified_c_D2', 'countdiag', 'countdiag_cat', 'C50_6d',
       'C64_C68_6d', 'Q00_Q99_x_H00_H59_6d', 'Q00_Q99_x_H60_H95_CH_6d',
       'V01_Y98_x_P00_P96_6d', 'C50_c_D2_6d', 'C64_C68_c_D2_6d',
       'L40_L45_c_D2_6d', 'M40_M54_c_D2_6d', 'N70_N77_c_D2_6d',
       'S00_S09_c_D2_6d', 'S80_S89_c_D2_6d', 'Z00_Z13_c_D2_6d',
       'U_Unclassified_c_D2_6d'],
      dtype='object')
```

Figure 11.2

In [8]:
```
# Create SelectKBest instance with the desired scoring function (e.g., f_classif for classification)
k_best_selector = SelectKBest(score_func=f_classif, k=50)  # Replace with your desired number of features

# Fit the selector to your data
k_best_selector.fit(X, Y)
```

Out[8]:
```
SelectKBest(k=50)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [9]:
```
# Transform the feature matrix to keep only the selected k best features
X_selected = k_best_selector.transform(X)

# Get a boolean mask of selected features
selected_mask = k_best_selector.get_support()

# Indexes of the selected features
selected_feature_indices = [i for i, selected in enumerate(selected_mask) if selected]

# Create a new DataFrame with the selected features
X_selected_df = X.iloc[:, selected_feature_indices]

# Print the column names of the selected features
selected_feature_columns = X_selected_df.columns
print("Selected feature columns:", selected_feature_columns)
```

```
Selected feature columns: Index(['C50', 'C64_C68', 'I00_I99', 'Q00_Q99_x_H00_H59', 'V01_Y98_x_P00_P96',
       'Z00_Z99_x_I00_I99', 'C50_c_D2', 'C64_C68_c_D2', 'H60_H95_c_D2',
       'K90_K93_c_D2', 'M40_M54_c_D2', 'N25_N29_c_D2', 'N70_N77_c_D2',
       'S00_S09_c_D2', 'S50_S59_c_D2', 'S80_S89_c_D2', 'Z00_Z13_c_D2',
       'U_Unclassified_c_D2', 'asp0', 'asp1', 'asp2', 'asp3', 'asp4', 'asp5',
       'asp6', 'asp7', 'asp8', 'asp9', 'asp10', 'asp11', 'countdiag',
       'countdiag_cat', 'C50_6d', 'C64_C68_6d', 'I00_I99_6d',
       'Q00_Q99_x_H00_H59_6d', 'Q00_Q99_x_H60_H95_CH_6d',
       'V01_Y98_x_P00_P96_6d', 'C50_c_D2_6d', 'C64_C68_c_D2_6d',
       'K40_K46_c_D2_6d', 'L40_L45_c_D2_6d', 'M40_M54_c_D2_6d',
       'N25_N29_c_D2_6d', 'N70_N77_c_D2_6d', 'S00_S09_c_D2_6d',
       'S50_S59_c_D2_6d', 'S80_S89_c_D2_6d', 'Z00_Z13_c_D2_6d',
```

Figure 11.3

In [10]:
```
# Create SelectKBest instance with the desired scoring function (e.g., f_classif for classification)
k_best_selector = SelectKBest(score_func=f_classif, k=70)  # Replace with your desired number of features

# Fit the selector to your data
k_best_selector.fit(X, Y)
```

Out[10]:
```
SelectKBest(k=70)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [11]:
```
# Transform the feature matrix to keep only the selected k best features
X_selected = k_best_selector.transform(X)

# Get a boolean mask of selected features
selected_mask = k_best_selector.get_support()

# Indexes of the selected features
selected_feature_indices = [i for i, selected in enumerate(selected_mask) if selected]

# Create a new DataFrame with the selected features
X_selected_df = X.iloc[:, selected_feature_indices]

# Print the column names of the selected features
selected_feature_columns = X_selected_df.columns
print("Selected feature columns:", selected_feature_columns)
```

```
Selected feature columns: Index(['A15_A19', 'C50', 'C64_C68', 'Z80_Z99', 'I00_I99', 'Q00_Q99_x_H00_H59',
       'Q00_Q99_x_L00_L99', 'V01_Y98_x_P00_P96', 'V01_Y98_x_Q00_Q99',
       'Z00_Z99_x_I00_I99', 'A20_A49_c_D2', 'C50_c_D2', 'C64_C68_c_D2',
       'H60_H95_c_D2', 'K40_K46_c_D2', 'K90_K93_c_D2', 'L40_L45_c_D2',
       'M40_M54_c_D2', 'N25_N29_c_D2', 'N70_N77_c_D2', 'Q90_Q99_c_D2',
       'R20_R23_c_D2', 'S00_S09_c_D2', 'S50_S59_c_D2', 'S80_S89_c_D2',
       'Z00_Z13_c_D2', 'U_Unclassified_c_D2', 'asp0', 'asp1', 'asp2', 'asp3',
       'asp4', 'asp5', 'asp6', 'asp7', 'asp8', 'asp9', 'asp10', 'asp11',
       'asp12', 'countdiag', 'countdiag_cat', 'countdiag1', 'A15_A19_6d',
       'C50_6d', 'C64_C68_6d', 'I00_I99_6d', 'Q00_Q99_x_H00_H59_6d',
       'Q00_Q99_x_H60_H95_CH_6d', 'Q00_Q99_x_L00_L99_6d',
       'V01_Y98_x_P00_P96_6d', 'V01_Y98_x_Q00_Q99_6d', 'Z00_Z99_x_I00_I99_6d',
```

Figure 11.4

Variance Threshold (Figure 12.1 – 12.2)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np

df = pd.read_csv("Norm_All_Float_NoMissing_NoTC_NoCF.csv")

X = df.drop(columns=['total_cost'])
y = df['total_cost']
```

In [2]:
```
from sklearn.feature_selection import VarianceThreshold
```

In [3]:
```
# Create VarianceThreshold instance
variance_threshold = VarianceThreshold(threshold=0.05)

# Fit the threshold to your data
variance_threshold.fit(X)
```

Out[3]:
```
VarianceThreshold(threshold=0.05)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [4]:
```
# Transform the feature matrix to keep only features above the threshold
X_high_variance = variance_threshold.transform(X)
```

In [5]:
```
# Get a boolean mask of selected features
selected_mask = variance_threshold.get_support()

# Indexes of the selected features
selected_feature_indices = [i for i, selected in enumerate(selected_mask) if selected]
```

Figure 12.1

In [6]:
```
# Get the column names of the selected features
selected_feature_names = X.columns[selected_feature_indices]

# Print the selected feature names
print("Selected feature names:", selected_feature_names)
```

```
Selected feature names: Index(['BB_5009_Pseudo_IDN', 'LSOA_', 'GP_CODE', 'male', 'CCGname_encode',
       'GPwProportionheadcountGPsfemale', 'GPwProportionofGPsaged50andov',
       'GPwProportionheadcountGPsquali', 'CouncilTaxBandAB',
       'CouncilTaxBandABC', 'WhiteEnglishWelshScottish', 'XXX', 'Z80_Z99',
       'Z00_Z99', 'asp3', 'asp4', 'asp5', 'countdiag1', 'r', 's1', 's2',
       'WhiteBritish', 'XXX_6d', 'Z80_Z99_6d', 'Z00_Z99_6d', 'countdiag6d1',
       'coastal', 'case_id', 'q05', 'AGE_a10_14', 'AGE_a15_19', 'AGE_a20_24',
       'AGE_a25_29', 'AGE_a30_34', 'AGE_a35_39', 'AGE_a40_44', 'AGE_a45_49',
       'AGE_a50_54', 'AGE_a55_59', 'AGE_a5_9', 'HH_type_Multi-adult',
       'HH_type_Multi-adult-child', 'HH_type_Single parent',
       'HH_type_Single person', 'HH_type_Two adult family',
       'HH_type_Two adults diff gender'],
      dtype='object')
```

Figure 12.2

Individual ML Algorithms (Figure 13.1 – 15.3)

Linear Regression (Figure 13.1)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np

df = pd.read_csv("LassoRegularisation_SubSample-Copy1.csv")

from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score, mean_squared_log_error

X = df.drop(columns=["total_cost"])
Y = df["total_cost"]
```

In [2]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5, random_state=42)
```

In [3]:
```
model = LinearRegression()
model.fit(X_train, Y_train)
```

Out[3]:
```
LinearRegression()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [4]:
```
Y_pred = model.predict(X_test)

mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
explained_variance = explained_variance_score(Y_test, Y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("MAE:", mae)
print("Explained Variance Score:", explained_variance)
```

Figure 13.1

Neural Network Regression (Figure 14.1 – 14.2)

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score, mean_squared_log_error
```

In [2]:
```
df = pd.read_csv("VarianceThreshold_SubSample-Copy1.csv")
X = df.drop(columns=["total_cost"])
Y = df["total_cost"]

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5, random_state=42)
```

In [3]:
```
# Standardize features (optional but recommended)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [4]:
```
model = MLPRegressor()
```

In [5]:
```
param_grid = {
    'hidden_layer_sizes': [(4,), (8,), (16,), (32,), (64,)]
}
```

In [6]:
```
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
grid_search.fit(X_train_scaled, Y_train)
```

Figure 14.1

In [7]:
```
best_model = grid_search.best_estimator_
Y_pred = best_model.predict(X_test_scaled)
```

In [8]:
```
# Evaluate the best model
mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
explained_variance = explained_variance_score(Y_test, Y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("MAE:", mae)
print("Explained Variance Score:", explained_variance)
```

```
Mean Squared Error: 17366.601442164258
R-squared: 0.058679755339368755
MAE: 94.14666445611209
Explained Variance Score: 0.058805240688900806
```

In [9]:
```
# Access best parameters from the grid search
best_hidden_layer_sizes = grid_search.best_params_['hidden_layer_sizes']

# Print the best values
print("Best 'hidden_layer_sizes':", best_hidden_layer_sizes)
```

```
Best 'hidden_layer_sizes': (64,)
```

Figure 14.2

Decision Tree (Figure 15.1 – 15.3)

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score, mean_squared_log_error

df = pd.read_csv("VarianceThreshold_SubSample-Copy1.csv")

X = df.drop(columns=["total_cost"])
Y = df["total_cost"]
```

In [2]:
```
# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.5, random_state=42)
```

In [3]:
```
param_grid = {
    'max_depth': [None, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
}
```

Figure 15.1

In [4]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import GridSearchCV

# Create a Decision Tree Regressor
decision_tree_model = DecisionTreeRegressor()

# Create GridSearchCV object
grid_search = GridSearchCV(estimator=decision_tree_model, param_grid=param_grid, cv=5)

# Perform grid search on training data
grid_search.fit(X_train, Y_train)

# Get the best model from grid search results
best_model = grid_search.best_estimator_

# Use the best model for predictions
Y_pred = best_model.predict(X_test)

# Evaluate the best model
mse = mean_squared_error(Y_test, Y_pred)
r2 = r2_score(Y_test, Y_pred)
mae = mean_absolute_error(Y_test, Y_pred)
explained_variance = explained_variance_score(Y_test, Y_pred)

print("Mean Squared Error:", mse)
print("R-squared:", r2)
print("MAE:", mae)
print("Explained Variance Score:", explained_variance)
```

```
Mean Squared Error: 17102.490771039866
R-squared: 0.07299532090285832
MAE: 92.06900594704469
Explained Variance Score: 0.07301764983815406
```

Figure 15.2

In [5]:
```
# Get the best max_depth value from grid search results
best_max_depth = grid_search.best_params_['max_depth']
print(best_max_depth)
```

```
4
```

Figure 15.3

Ensemble ML Algorithms (Figure 16.1 – 19.2)

Bagging Regressor (Figure 16.1 – 16.2)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np
```

In [2]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score, mean_squared_log_error
```

In [3]:
```
df = pd.read_csv("Norm_All_Float_NoMissing_NoTC_NoCF-Copy1.csv")
```

In [4]:
```
X = df.drop(columns=["total_cost"])
Y = df["total_cost"]
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

In [6]:
```
base_estimator = DecisionTreeRegressor()

# Create a BaggingRegressor model
bagging_regressor = BaggingRegressor(base_estimator=base_estimator)

param_grid = {
    'n_estimators': [50, 100, 200, 500, 1000], # Number of base estimators in the ensemble.
    'max_samples' : [0.5, 0.8, 1.0],
}
```

In [ ]:
```
grid_search = GridSearchCV(estimator=bagging_regressor, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)  # X_train and y_train are your training data.
```

Figure 16.1

```
In [ ]: best_params = grid_search.best_params_
        best_model = grid_search.best_estimator_
```

```
In [ ]: # View the best parameters
        print("Best Parameters:")
        print(best_params)
```

```
In [ ]: # Assuming X_test and y_test are your test data
        y_pred = best_model.predict(X_test)

        mse = mean_squared_error(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)

        print("Mean Squared Error:", mse)
        print("Mean Absolute Error:", mae)
        print("R-squared:", r2)
```

Figure 16.2

CatBoost Regressor (Figure 17.1 – 17.2)

```
In [1]: import pandas as pd
        import tensorflow as tf
        import numpy as np
        from catboost import CatBoostClassifier, CatBoostRegressor
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score, mean_squared_log_error
        from sklearn.model_selection import GridSearchCV
```

```
In [2]: df = pd.read_csv("KBest_SubSample_70-Copy1.csv")
```

```
In [3]: # Separate features and target variable
        X = df.drop(columns=["total_cost"])
        y = df["total_cost"]

        # Split data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
In [4]: param_grid = {
            'iterations': [2000],  # Number of boosting iterations
            'depth': [2, 3, 4, 5],            # Maximum depth of trees
            'learning_rate': [0.015, 0.01, 0.0075],  # Learning rate
            'n_estimators' : [200, 500, 1000]   # Number of estimators
            'loss_function': ['RMSE']
        }
```

```
In [5]: # For Regression
        catboost_model = CatBoostRegressor()

        # Create a GridSearchCV object
        grid_search = GridSearchCV(estimator=catboost_model, param_grid=param_grid, scoring='r2', cv=5)

        # Fit the grid search to your data
        grid_search.fit(X_train, y_train)

        # Get the best parameters and best estimator
        best_params = grid_search.best_params_
        best_model = grid_search.best_estimator_
```

Figure 17.1

```
In [6]: best_params = grid_search.best_params_
        print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'depth': 3, 'iterations': 2000, 'learning_rate': 0.0075, 'loss_function': 'RMSE'}
```

```
In [7]: # Assuming X_test and y_test are your test data
        y_pred = best_model.predict(X_test)

        mse = mean_squared_error(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)

        print("Mean Squared Error:", mse)
        print("Mean Absolute Error:", mae)
        print("R-squared:", r2)
```

```
Mean Squared Error: 16862.942650057317
Mean Absolute Error: 94.3797712622581
R-squared: 0.05802467229495434
```

Figure 17.2

LightGBM (Figure 18.1 – 18.2)

In [1]:
```
import pandas as pd
import tensorflow as tf
import numpy as np
```

In [2]:
```
from sklearn.model_selection import train_test_split, GridSearchCV
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score, mean_squared_log_error
```

In [3]:
```
df = pd.read_csv("RandomForest_SubSample_70-Copy1.csv")
```

In [4]:
```
X = df.drop(columns=["total_cost"])
Y = df["total_cost"]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
```

In [5]:
```
param_grid = {
    'n_estimators': [250, 500, 1000],
    'learning_rate': [0.01],
    'max_depth': [5, 10, 15, 20],
    'num_leaves': [31, 63, 90, 120]
    # Add more hyperparameters as needed
}
```

In [6]:
```
# Create an LGBMRegressor model
lgb_regressor = LGBMRegressor(random_state=42)

grid_search = GridSearchCV(estimator=lgb_regressor, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1, n_j

# Train the model
grid_search.fit(X, Y)
```

Figure 18.1

In [7]:
```
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_
```

In [8]:
```
# View the best parameters
print("Best Parameters:")
print(best_params)
```

```
Best Parameters:
{'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 250, 'num_leaves': 63}
```

In [9]:
```
# Assuming X_test and y_test are your test data
y_pred = best_model.predict(X_test)

mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Mean Squared Error:", mse)
print("Mean Absolute Error:", mae)
print("R-squared:", r2)
```

```
Mean Squared Error: 15683.446378639457
Mean Absolute Error: 89.04410028953465
R-squared: 0.12391212799308182
```

Figure 18.2

Random Forest (Figure 19.1 – 19.2)

```
In [1]: import pandas as pd
        import tensorflow as tf
        import numpy as np
        from sklearn.model_selection import train_test_split, GridSearchCV
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, explained_variance_score, mean_squared_log_error

In [2]: df = pd.read_csv("GradientBoostingRegressor_SubSample_50-Copy1.csv")

In [3]: X = df.drop(columns=["total_cost"])
        Y = df["total_cost"]

        # Split the data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.5, random_state=42)

In [ ]: # Create a RandomForestRegressor model
        random_forest = RandomForestRegressor()

        param_grid = {
            'n_estimators': [100, 200, 500],  # Number of trees in the forest.
            'max_depth': [None, 5, 10, 15],  # Maximum depth of each tree.
        }

        grid_search = GridSearchCV(estimator=random_forest, param_grid=param_grid, cv=5)
        grid_search.fit(X_train, y_train)  # X_train and y_train are your training data.

In [ ]: # Make predictions
        y_pred = rf_model.predict(X_test)
```

Figure 19.1

```
In [ ]: mse = mean_squared_error(y_test, y_pred)
        r2 = r2_score(y_test, y_pred)
        mae = mean_absolute_error(y_test, y_pred)
        explained_variance = explained_variance_score(y_test, y_pred)

        print("Mean Squared Error:", mse)
        print("R-squared:", r2)
        print("MAE:", mae)
        print("Explained Variance Score:", explained_variance)
```

Figure 19.2

Hyperparameter Tuning (Figure 20.1 – 20.6)

Neural Network Regression (Figure 20.1)

```
In [5]: param_grid = {
            'hidden_layer_sizes': [(4,), (8,), (16,), (32,), (64,)]
        }
```

Figure 20.1

Decision Tree (Figure 20.2)

```
In [3]: param_grid = {
            'max_depth': [None, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
        }
```

Figure 20.2

Bagging Regressor (Figure 20.3)

```
param_grid = {
    'n_estimators': [50, 100, 200, 500, 1000], # Number of base estimators in the ensemble.
    'max_samples' : [0.5, 0.8, 1.0],
}
```

Figure 20.3

CatBoost Regressor (Figure 20.4)

In [4]:
```
param_grid = {
    'iterations': [2000],  # Number of boosting iterations
    'depth': [2, 3, 4, 5],            # Maximum depth of trees
    'learning_rate': [0.015, 0.01, 0.0075],  # Learning rate
    'n_estimators' : [200, 500, 1000]   # Number of estimators
    'loss_function': ['RMSE']
}
```

Figure 20.4

LightGBM (Figure 20.5)

In [5]:
```
param_grid = {
    'n_estimators': [250, 500, 1000],
    'learning_rate': [0.01],
    'max_depth': [5, 10, 15, 20],
    'num_leaves': [31, 63, 90, 120]
    # Add more hyperparameters as needed
}
```

Figure 20.5

Random Forest (Figure 20.6)

```
param_grid = {
    'n_estimators': [100, 200, 500],  # Number of trees in the forest.
    'max_depth': [None, 5, 10, 15],  # Maximum depth of each tree.
}
```

Figure 20.6