When Buildings Die

A Comparative Investigation of Machine Learning Approaches for Building Lifetime Modeling in Denmark

Authors:

Theodor Dornonville de la Cour, Linus Juni, and Oscar Johan Høeg Wohlfahrt

Department of Applied Mathematics and Computer Science

June, 2025

When Buildings Die
A Comparative Investigation of Machine Learning Approaches for Building Lifetime Modeling in Denmark

Bachelor's Thesis
June 2025

Theodor Dornonville de la Cour
Linus Juni
Oscar Johan Høeg Wohlfahrt

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark
Richard Petersens Plads, Building 324
2800 Kgs. Lyngby
Denmark

Formalities

We hereby declare that this bachelor's thesis has been written independently by the authors. All sources and references used have been properly cited and acknowledged. The work presented is original and has not been submitted for any other academic degree.

We have used artificial intelligence tools from OpenAI, Anthropic, and Google to assist with code, literature review, and writing support. In accordance with DTU's guidelines for documenting the use of Generative AI, we acknowledge this usage and can provide detailed documentation including prompts, AI responses, and version information upon request. All AI-generated content has been reviewed and verified. The core research methodology, analysis, and conclusions remain entirely our own work.

If you wish to access the code used in this project, please send us an email and we will add you to our GitLab repository hosted on DTU servers.

This thesis is submitted in partial fulfillment of the requirements for the BSc in General Engineering at the Technical University of Denmark.

Date: 13th June 2025
Location: Copenhagen, Denmark

Dornonville de la Cour, Theodor

Juni, Linus

Wohlfahrt, Oscar Johan Høeg

Abstract

Accurately predicting building lifetime is essential for urban planning and sustainable development, yet modern survival analysis methods have not been thoroughly evaluated on large-scale building datasets in Denmark. This proof-of-concept thesis investigates which survival analysis methods show the most predictive potential for Danish building data using the Danish Building and Housing Register (BBR), containing useful information on 4.4 million unique buildings.

Using nested cross-validation we compare traditional methods (Kaplan-Meier, Nelson-Aalen, Cox Proportional Hazards), ensemble methods (Random Survival Forest, Gradient Boosting, and Component-wise Gradient Boosting), and a deep learning approach (DeepSurv) alongside a Benchmark model that combines feature-specific Kaplan-Meier estimators. Rather than predicting exact lifetimes, we evaluate each method's ability to correctly rank buildings by lifetime using Harrell's concordance index (C-index).

Despite notable data challenges, we achieve reasonable C-index values. Computational limitations necessitate training models on varying sample sizes, therefore subsets ranging from 0.1% to 100% are sampled with stratification from the complete dataset. Across most evaluations, traditional and ensemble methods demonstrate consistent performance without statistically significant differences, while DeepSurv shows highest potential but suffers from training instability. Feature importance analysis generally identifies physical building characteristics as the most significant predictors of demolition risk, whereas geographic location provides minimal predictive value.

This thesis establishes the feasibility of applying survival analysis methods to building lifetime prediction while highlighting the need for improved data quality in future research.

Acknowledgements

We would like to thank our supervisor, Nicolai Siim Larsen, for his excellent guidance and support throughout this bachelor project. Nicolai consistently went above and beyond, meeting with us weekly and providing valuable feedback that helped shape our work. We are grateful for his dedication to our success and for making this challenging project both manageable and rewarding.

We thank the Technical University of Denmark and the Department of Applied Mathematics and Computer Science for providing the resources and academic environment that made this thesis possible.

We would also like to thank each other. We had great discussions, pushed each other hard, and somehow remained great friends through it all. This thesis was like the IKEA test to the extreme. We're proud of both the result and our friendship that survived it.

Special thanks go to Sebastian Pölsterl for developing scikit-survival and Håvard Kvamme for creating pycox – essential tools for our thesis.

We thank our friends and families for their support and encouragement, not only during this thesis but throughout our entire three years of bachelor studies.

Finally, we acknowledge the coffee machine in Building 324, without which this thesis would have been significantly less caffeinated and considerably more challenging to complete.

Notation

- C_i: The construction year for building i
- D_i: The demolition year for building i
- t_i: Time index (in whole years), $t_i \in \{0, 1, 2, ...\}$, where t_i is indexed from C_i for building i. In other words, the age of building i
- T: Discrete random variable representing the building's lifetime (in whole years) from construction until demolition, $T \in \{0, 1, 2, ...\}$
- Y_i: The observed lifetime for building i. This is the duration (in whole years) from its construction until demolition or censoring, $Y_i \in \{0, 1, 2, ...\}$
- $f(t)$: Probability mass function, $f(t) = P(T = t)$
- $F(t)$: Cumulative distribution function, $F(t) = P(T \leq t)$
- $S(t)$: Survival function, $S(t) = P(T > t) = 1 - F(t)$
- $h(t)$: Hazard function, $h(t) = P(T = t \mid T \geq t)$
- $H(t)$: Cumulative hazard function, $H(t) = \sum_{k=0}^{t} h(k)$
- d_t: Number of demolitions at t
- n_t: Number of subjects at risk just prior to t
- $I(\cdot)$: Indicator function, equals 1 if its argument is true, 0 otherwise
- δ_i: Event indicator for building i
  - $\delta_i = 1$ if demolition time is observed
  - $\delta_i = 0$ if right-censored

In the following, a hat ($\hat{\cdot}$) above a function or parameter will denote its estimate. For example, $\hat{S}(t)$ represents the estimated survival function, $\hat{h}(t)$ represents the estimated hazard function, and $\hat{H}(t)$ represents the estimated cumulative hazard function.

Contents

1 Introduction

As cities grow and age, the question of when to replace existing buildings becomes increasingly important. Accurately predicting building lifetime – the time from construction to demolition – is essential for urban planners, building owners, and other stakeholders. With reliable lifetime estimates, better decisions can be made about materials, construction methods, and maintenance.

Getting these predictions wrong has consequences. Underestimating may lead to unnecessary early replacements, wasting resources, and harming the environment. Overestimating can result in inadequate maintenance and potential building failures. For sustainable development, accurate lifetime predictions are vital for assessing environmental impacts over a building's entire lifecycle.

This thesis serves as a proof-of-concept study exploring the potential of modern survival analysis methods for building lifetime estimation. We implement and compare four methodological categories: traditional non-parametric methods (Kaplan-Meier and Nelson-Aalen estimators), regression-based approaches (Cox Proportional Hazards model), ensemble methods (Random Survival Forests and Gradient Boosting), deep learning techniques (DeepSurv), and introduce a novel benchmark method. Rather than aiming for definitive lifetime predictions, our primary goal is to demonstrate the feasibility of these approaches when applied to real-world building data.

The data to support this analysis comes from the Danish Building and Housing Register (BBR), a centralized national database that has systematically recorded information on buildings since 1976. While the BBR provides a good foundation for modeling lifetime, it is not without challenges. The data collection method, as well as which features are recorded, has changed multiple times throughout the years, leading to data issues such as unrealiable features. Furthermore, inconsistencies in how demolitions are reported and a lack of clear documentation thereof introduce systematic biases. The data's inherent flaws limit the generalizability of conclusions, so carefully interpreting model outputs is important.

As a proof-of-concept, this work seeks to help answer three fundamental questions: which survival analysis methods work best for Danish building data, how certain building characteristics impact the predicted lifetime of buildings, and how Danish building data can be used for model training and prediction in survival analysis. Our findings reveal some of the strengths and weaknesses of each modeling approach, creating a starting point for future researchers working on building lifetime prediction. In essence, this thesis lays the foundation for more advanced research rather than providing ready-to-use prediction tools.

2 Background and Literature Review

Before diving into the technical details of survival analysis and machine learning methods, it's helpful to understand what types of buildings and data we're actually working with in this thesis.

The Buildings We Study Our analysis draws from the Danish Building and Housing Register (BBR). BBR is a nationwide register in Denmark that contains detailed information about buildings, dwellings, and other installations. It includes data such as construction year, size, use, materials, heating sources, and renovation history. It gives us access to information about 6 million individual buildings (whereof about 4.4 million are actually useful) across the entire country – from historic timber-framed farmhouses built in the 1400s to modern glass offices completed this year.

Construction and demolition years range from 1426 up to and including 2024, which is the cutoff year for this study. The vast majority of buildings are constructed in newer times, especially after World War II. When we examine building demolitions, we see a much more concentrated pattern. Most recorded demolitions occurred between 2000 and 2024, however only with reliable data available from 2016 onwards. This concentration does not reflect an absence of demolitions prior to 2000, but rather the limitations of the data collection in earlier periods.

Right-censoring dominates our dataset: of the 4.4 million buildings, only about 250,000 (5.7%) have recorded demolition dates. Right-censoring means demolition hasn't occurred by the cutoff. The remaining 4.2 million buildings (94.3%) are still standing as of our data collection cutoff. While this is a limitation, it is also expected in survival analysis, and thus, survival analysis methods are specifically designed to incorporate both complete and censored observations into parameter estimation and risk prediction.

Importantly, censoring is not missing data. It's partial information that contains real value. Knowing that a building has lasted for 60 years and is still standing tells us precisely that its time-to-demolition exceeds 60 years, and survival models can use that fact to learn and improve prediction.

The following sections present the theoretical foundations of survival analysis as applied to building demolition modeling. We establish the mathematical framework for discrete-time survival models, describe the nature of censored observations, and review existing literature on survival analysis applications to building service life prediction, identifying methodological approaches and empirical findings relevant to our analysis.

2.1 Fundamentals of Survival Analysis

Survival analysis is a branch of statistics that estimates the *time* T until an *event* E occurs. An event could be the death of a person, finishing a degree, or, in our case, the demolition of a building. The lifetime of each subject begins at a well-defined origin, such as birth, enrollment, or the day a building is built (represented by ○ in Figure 1). In this thesis T is measured in discrete units (specifically, whole years). These individual time-to-event histories can be either fully observed or only partly observed (censored), resulting in different types of data "profiles". Figure 1 shows the types of time-to-event profiles present in the project's dataset, including examples of both recorded demolitions and censored events.

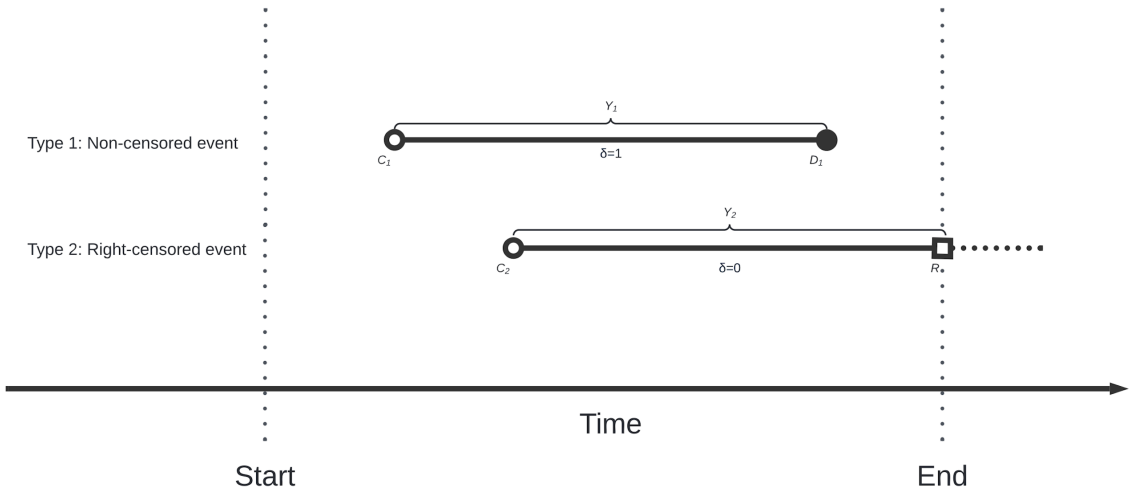

Figure 1: Time-to-event profile of the types of building lifetimes present in the project. ○ denotes the construction of a building, ● denotes the observed demolition of a building, while □ denotes a censored event.

The different scenarios shown are important for understanding real-world data. These will be explained in more detail when we cover censoring in Section 2.1.3.

2.1.1 Defining Survival: Core Functions

Survival analysis in the discrete domain relies on five key functions – namely the probability mass function (pmf), the cumulative distribution function (cdf), the survival function, the hazard function and the cumulative hazard function. Since we analyze building demolitions by year, we work exclusively with discrete time throughout this

thesis. We begin by defining the probability mass function, $f(t)$, and cumulative distribution function, $F(t)$:

$$f(t) = P(T = t), \quad F(t) = P(T \leq t), \tag{1}$$

where T is a discrete random variable representing the lifetime of a subject, with $T \in \{0, 1, 2, \ldots\}$. The survival function $S(t)$ measures the probability that an event has not occurred at the end of t:

$$S(t) = P(T > t), \tag{2}$$

and can equivalently be written in terms of $F(t)$:

$$S(t) = 1 - F(t).$$

By definition $S(t)$ is non-increasing. In the continuous time domain, $S(0) = 1$, but since we are working with discrete time, $S(0)$ is not necessarily equal to 1 as a building may have been torn down the same year it was built thus having a lifetime of 0 years. The most common way of estimating $S(t)$ is using the Kaplan-Meier estimator (see Section 5.1.1).

The hazard function $h(t)$ quantifies the proba of an event E happening at t, given that it has not occurred just prior to t. In other words, if a building has not been demolished just prior to t, how likely is it that it will be demolished at t. Formally,

$$h(t) = P(T = t \mid T \geq t). \tag{3}$$

The hazard at t can be estimated by,

$$\hat{h}(t) = \frac{d_t}{n_t},$$

where d_t is the number of events at t, and n_t is the number of subjects at risk just prior to t (i.e. buildings not demolished just prior to t). Accumulating the hazards $h(t)$ yields the cumulative hazard function $H(t)$, which keeps track of how much total hazard has accumulated by t:

$$H(t) = \sum_{k=0}^{t} h(k), \tag{4}$$

which is non-decreasing. The most common way of estimating $H(t)$ is using the Nelson-Aalen estimator (see Section 5.1.2).

2.1.2 Relationships Between Functions

The key functions in survival analysis, namely $f(t)$, $F(t)$, $S(t)$, $h(t)$, and $H(t)$ are all closely related. Having presented certain relations in the previous section, we proceed to discuss further relationships. We remind the reader that in this thesis we deal with discrete time, formally $t \in \{0, 1, 2, \ldots\}$. Thus, in the discrete case $h(t)$ can be written as

$$h(t) = \frac{f(t)}{S(t-1)}, \text{ where } S(-1) = S(0),$$

since $f(t)$ measures the probability of an event happening at t, and $S(t-1)$ measures the probability that an event has not occurred just prior to t.

Furthermore, in the discrete case there is also a relationship between $S(t)$ and $h(t)$

$$S(t) = \prod_{k=0}^{t} (1 - h(k)),$$

which is the foundation of the Kaplan-Meier estimator presented in Section 5.1.1.

Lastly, in addition to Equation 4, $H(t)$ is also related to $S(t)$ through the relation:

$$S(t) \approx e^{-H(t)}$$

In the continuous-time case, the relationship between survival and cumulative hazard becomes exact. The hazard function in the continuous case is defined as:

$$h(t) = \frac{f(t)}{S(t)},$$

where $f(t) = -\frac{d}{dt}S(t)$ is the probability density function (Breheny, 2015). Substituting gives:

$$h(t) = -\frac{1}{S(t)} \cdot \frac{d}{dt}S(t),$$

This is a separable differential equation, which can be written as:

$$\int \frac{1}{S(t)}\, dS(t) = -\int h(t)\, dt,$$

solving these integrals gives

$$\log S(t) = -H(t) + C.$$

Applying the initial conditions $S(0) = 1$ and $H(0) = 0$, we find $C = 0$, so:

$$S(t) = \exp(-H(t)).$$

This identity does *not* hold in discrete time because the survival and hazard functions are not continuous. We return to this distinction later in Section 5.1.3, where we discuss why both the Kaplan-Meier and Nelson-Aalen estimators are needed in practice.

Summary This finally leaves us with the discrete time relations presented in Table 1.

Function	Expression in terms of other functions
$\boldsymbol{S(t)}$	$S(t) = 1 - F(t)$
	$S(t) = \prod_{k=0}^{t}(1 - h(k))$
	$S(t) \approx e^{-H(t)}$
$\boldsymbol{h(t)}$	$h(t) = \dfrac{f(t)}{S(t-1)}$
$\boldsymbol{H(t)}$	$H(t) = \sum_{k=0}^{t} h(k)$

Table 1: Relationships between $S(t)$, $h(t)$, and $H(t)$ in discrete time

Other relations between these functions exist, but we will not touch on them in this thesis.

2.1.3 Censoring in Survival Analysis

A core part of survival analysis is that it accounts for censoring in the data. Censoring occurs when the event of interest is not observed. Throughout this thesis, the event of interest is *always* a demolition. The most common form of censoring is right-censoring, and it applies to the majority of buildings in our dataset. The following is based on notes from Lunn (2007), Oxford University, Department of Statistics.

Right-censoring Generally in survival analysis, right censoring occurs when a subject "leaves the study" before the event of interest occurs, or the study ends before the event has occurred. In the context of buildings, this happens when the observation period ends before a building is demolished. Thus, an observation is right-censored if

we know that the event has not occurred by the end of the right-censoring year, but do not observe exactly when it will occur. Formally, let T_i be the discrete random variable representing the true lifetime for building i, C_i be the construction year of building i, and R be the right-censoring year. For a right-censored observation, we observe

$$Y_i = \min(T_i, R - C_i) = R - C_i \tag{5}$$
$$\delta_i = 0, \tag{6}$$

where Y_i is the observed lifetime for building i and δ_i is the censoring indicator. Importantly, for right-censored observations, Y_i represents the *minimum amount of years we know the building survived* – it tells us how long we observed the building standing, not when it will actually be demolished. We know only that $T_i > (R - C_i)$. For example, if a building constructed in 1900 is still standing at the end of the last year before the writing of this thesis, 2024 (as illustrated by Type 2 in Figure 1), its observed lifetime is $Y_i = 2024 - 1900 = 124$ years with $\delta_i = 0$, and we know only that the true lifetime $T_i > 124$.

Non-censoring When a subject experiences the event of interest within the observation period, it is considered non-censored. In our case, a building is considered non-censored, when the demolition falls within the observation period (illustrated as Type 1 in Figure 1). Such a subject's observed lifetime is given by:

$$Y_i = \min(T_i, R - C_i) = T_i \tag{7}$$
$$\delta_i = 1 \tag{8}$$

In this case, we observe the true lifetime directly, so $Y_i = T_i$ and $T_i \leq (R - C_i)$.

Left-censoring Another form of censoring is left-censoring. Generally, left censoring occurs when the event of interest has already occurred before enrollment in the study. This is very rarely encountered in practice. In the building context, this would occur if we knew a building had been demolished sometime before our records began, but we did not know the exact demolition time, but we did observe the construction year. We do not observe left-censored data in this thesis, and thus, we will not introduce it formally.

However, a related phenomenon that we do observe, known as left truncation, is when subjects have been at risk before entering the study. In our building dataset, this occurs because we only observe buildings that survived until the beginning of record-keeping in BBR, creating survivorship bias as discussed in Section 4.3.2.

Summary For each building i, we observe (Y_i, δ_i), where:

- $\delta_i = 1$: $Y_i = T_i$ (demolition observed, $T_i \leq R - C_i$).
- $\delta_i = 0$: $Y_i = R - C_i$ (right-censored, $T_i > R - C_i$).

In essence, right censored observations provide bounds on the true lifetime, because it gives us a lower bound ($T_i > Y_i$). Only when $\delta_i = 1$ do we observe the exact lifetime ($T_i = Y_i$).

In this thesis, we focus exclusively on right-censored and true observed data, since we do not observe left-censored data, so our dataset contains only observations with $\delta_i \in \{1, 0\}$. This will become very useful later, when we focus on model training, since they are specifically designed for right-censoring.

2.2 Literature Review

In this section, we provide an overview of methods previously used to estimate the service life or survival time of buildings. While some studies focus on Denmark, we also look into relevant international research to give a broader view of the methods used. We will dive deeper into some of the models presented in this review and investigate the performance of on Danish building data.

Statistical Approaches to Service Life Estimation One of the most widely used methods in the building sector is the Factor Method. The method was presented in ISO (2011) and has gained popularity for its simple and practical approach to estimating life times. The method estimates the service life of building components using a reference service life (RSL) and adjusting it with modifying factors. These factors represent specific conditions that influence the service life of a component, for instance, construction quality, design and maintanance level. The estimated service life is computed as the product of the RSL and the modifying factors. While easy to use, the method relies on externally determined RSLs and factor values, which must be calibrated through other scientific or empirical means. For example, standard RSL values can vary significantly depending on the source and application. Jordan (2019) showed that dependent on the selected RSL database, the results can differ by up to a factor of five. The Factor Method is further complicated by the subjective assignment of modifying factors, which, as Francart, Widström, and Malmqvist (2021) showed, can lead to significantly varying service life estimates. In addition, the Factor Method does not yield any probability distribution or survival curve, only a single fixed lifetime estimate. Despite these limitations, the Factor Method remains popular in practice

because it provides a structured approach that is familiar to practitioners and requires limited data input (Amemiya and Yalcin, 2001).

Another common approach, particularly in economic modeling, is to assume a constant annual demolition rate. For instance, the Annual Danish Aggregate Model (ADAM), a model for the Danish economy used by the Danish Ministry of Finance (2024), assumes a 1% demolition rate. This geometric model with mean $E = 1/p$ (Ross, 2014) implies a 100-year mean lifespan. Aagaard et al. (2013) have proposed a model that is potentially more true to life, assuming a 20-30 year demolition free period, followed by a demolition rate of 0.3%, resulting in a mean service life between 353 and 363 years under a geometric distribution. These approaches, while informed by BBR data in Aagaard's case, employ simplified models rather than survival analysis methods that can account for censoring, and building characteristics.

Data Driven Methods for Survival Time In recent years, the emergence of large-scale building datasets has enabled more data-driven approaches to estimating survival times. While high-quality, representative data remains rare, national registries such as the Danish Building and Housing Register (BBR)* provide a good starting point for analysis.

Andersen and Negendahl (2023) use demolition data from BBR (2010- 2017) to estimate building service life. They found service life to be decreasing as a function of construction year. However, follow-up work, for instance by Jensen et al. (2024), has highlighted bias and censoring issues, noting that such studies may suffer from survivorship bias and limited observation windows. To overcome this, newer analyses by Droob and Nybroe (2024) have introduced left-censored data and simulation-based backcasting, resulting in more robust survival estimates. Challenges do however still remain in accurately reconstructing historical demolition data.

Survival Models and Machine Learning Cox Proportional Hazards (Cox PH) models are widely used in survival analysis to estimate the influence of covariates on the hazard of an event (e.g., demolition). Shima et al. (2003) applied this approach in Japan to predict demolition waste generation, showing how economic and structural variables influence building longevity.

Building on this, DeepSurv was introduced by Katzman et al. (2018). DeepSurv is a deep learning generalization of the Cox model, it replaces the linear component of the model with a neural network, to capture non-linear risk patterns. Although DeepSurv has primarily been used in healthcare, such as in modeling survival times for breast cancer patients, it has yet to be widely adopted in the building sector.

*Introduced in Section 4.1.1

Ensemble tree methods have also gained traction. Random Survival Forests (RSF), introduced by Ishwaran et al. (2008), extend random forests to handle censored survival data. RSF models can capture complex, non-linear interactions without requiring parametric assumptions. While RSF has not yet been applied to Danish building data, it has been used to model deterioration of infrastructure, such as bridge decks, where it outperformed traditional models like the Weibull-based Accelerated Failure Time model (Lu and Ilgin Guler, 2022).

Boosting methods, such as XGBoost and LightGBM, have also been used for service life prediction. A notable example is a South Korean study by Ji, Lee, and Yi (2021), which applied boosting and deep learning models to a national dataset of buildings. Their models achieved high predictive performance (R^2 between 0.92 and 0.95). However, the authors noted concerns with data quality and bias. In addition, the findings may not generalize to other contexts such as Denmark, due to differences in construction practices and building regulations.

Overall, data-driven survival modeling in the building sector remains underexplored, largely due to the lack of comprehensive, high-quality datasets. Additionally, the problem is often framed through a civil engineering lens rather than a data science perspective. Our thesis contributes to this area by applying modern survival models (Cox PH, RSF, DeepSurv, and gradient boosting) to Danish demolition data. Our aim is to assess and compare the predictive value of these models for building service life estimation. We do however want to emphasize that our analysis serves as a proof of concept, because of limitations in data and computational resources, discussed further in later sections.

3 Methodology

The literature review shows several important patterns in building service life prediction. Traditional methods like the Factor Method are easy to use but depend on parameters determined outside the model. Economic models often assume that buildings are demolished at a constant rate each year, which doesn't match reality. Furthermore, our review reveals that modern survival analysis methods – while successful in healthcare and other fields – haven't been thoroughly tested for predicting building lifespans using large building registry datasets.

This gap, combined with the comprehensiveness of the Danish Building and Housing Register (BBR), creates both an opportunity and a challenge. Unlike many countries that lack centralized, long-term, and detailed building data, the BBR provides a great scale of information on Denmark's entire building stock (Agency for Climate Data, 2025). However, as we will show in Section 4, it was not originally designed for survival analysis and has built-in limitations that affect how we train and evaluate models.

Our Approach and Goals This section outlines our approach for comparing different survival analysis methods using the data at hand. Rather than trying to make exact lifetime predictions – which the data quality issues make unreliable – our thesis is designed as a proof-of-concept study with three main goals as stated in the Introduction:

- Investigate which survival analysis methods work best for building data
- Figure out how Danish building data can be used for model training and prediction in survival analysis
- Determine how certain building characteristics impact the predicted lifetime of buildings

To achieve these goals, we will evaluate three categories of survival analysis methods. We start with simple approaches and gradually move to more complex machine learning techniques. This allows us to see how added complexity affects performance when working with building data.

Traditional Statistical Methods serve as our starting point. These include the Kaplan-Meier estimator for calculating survival probabilities, the Nelson-Aalen estimator for measuring cumulative risk, and the Cox Proportional Hazards model for understanding how building features affect demolition risk. These methods have been used successfully in many fields and provide clear, interpretable results.

Ensemble Methods combine multiple models to make better predictions. Random Survival Forests and Gradient Boosting can find complex patterns and interactions between building characteristics that traditional methods might miss. These approaches have been shown to perform well in other sectors.

Deep Learning Methods use neural networks to find relationships in the data. We implement DeepSurv, which extends the Cox model with neural networks. While this methods is harder to interpret, it has been shown to perform well. Furthermore, given the growing success of deep learning across various domains, including survival analysis applications in healthcare, we include this approach to assess its potential for building service life prediction.

By comparing these methods systematically, from simple to sophisticated, we can try to determine which approaches work best with the building data we have. We also learn about the trade-offs between model complexity, how easy results are to interpret, and how well the models predict outcomes.

It should be noted that the results we obtaine will be heavily influenced by the bias and low qualuty of our dataset. This will be discussed in futher detail in Section 4.3.

The following methodology section outlines the evaluation framework we use consistently across all three types of methods.

3.1 Model Evaluation

In this section, we introduce the methods used to compare the different statistical and machine learning models. First, we introduce a performance metric, which is calculated for each model and used for direct comparison. Second, we examine feature importance to understand which input variables drive predictions and how their influences vary across models. Lastly, we discuss cross-validation across all models to ensure robustness and increase the generalizability of the results.

Essentially, this evaluation framework seeks to answer three key questions: How do we measure success when dealing with heavily censored survival data? How do we understand which building characteristics matter most across different models? And how do we ensure our performances and comparisons are robust?

3.1.1 Performance Metrics

Evaluating and comparing survival models requires performance metrics that are suited for handling censored data, where the exact event time is unknown for some subjects, as presented in Section 2.1.3. Standard regression metrics such as mean squared error are unsuitable for survival analysis because they cannot properly account for censored observations and the uncertainty in survival times this leads to.

Several metrics have been developed for survival analysis, with the concordance index (C-index) being the most widely used. This metric forms the basis for model evaluation in our thesis and will be presented in the following section.

Concordance Index Harrell et al. (1996) introduce the Harrell's concordance index (C-index) as the proportion of all comparable pairs in which the predictions and outcomes are concordant. Two observations are comparable if either:

- Both experienced an event, and the events occurred at different times
- The observation with the shorter observed survival time experienced an event, while the other remained event-free – meaning the latter "outlived" the former.

Pairs in which both observations experienced events at the same time are not considered comparable.

For a pair of observations to be concordant, the model's risk predictions[†] must align with the observed survival outcomes. Specifically, two observations (i, i') form a concordant pair if the observation with the higher predicted risk score actually experiences an earlier event time. In other words, if the model assigns a higher risk[‡] to observation i' than to observation i ($\hat{\eta}_{i'} > \hat{\eta}_i$), and observation i' indeed has a shorter survival time ($Y_{i'} < Y_i$), then this pair is concordant.

The C-index intuitively reflects the model's ability to correctly rank pairs of observations based on their predicted risk. Formally, the C-index is calculated as the ratio of concordant pairs to comparable pairs:

$$C = \frac{\text{Concordant pairs}}{\text{Comparable pairs}} \tag{9}$$

To compute this, we first count the comparable pairs (denominator). As by the definition of a comparable pair above, any pair of observations (i, i') where $Y_i > Y_{i'}$ is comparable if the observation i' (with shorter survival time) experienced the event (i.e., $\delta_{i'} = 1$).

This ensures we only consider pairs where either both experienced events at different times, or where the shorter-lived observation experienced an event while the longer-lived one may still be censored. The number of comparable pairs is therefore:

[†]Risk predictions are introduced in section Section 6.1.1. They quantify the likelihood of a building getting demolished such that higher risk scores correlate with earlier demolitions.

[‡]Which we can also denote as Risk(x_i), but for the purpose of demonstrating the Concordance Index, we prefer the shorter notation of $\hat{\eta}_i$

$$\text{Comparable pairs} = \sum_{i,i':Y_i>Y_{i'}} \delta_{i'} \tag{10}$$

For the numerator, we assign a concordance score to each comparable pair:

$$\text{Concordance score} = I(\hat{\eta}_{i'} > \hat{\eta}_i) + \frac{1}{2} I(\hat{\eta}_{i'} = \hat{\eta}_i) \tag{11}$$

where $\hat{\eta}_i$ is the predicted risk score for observation i, and $I(\cdot)$ is the indicator function that equals 1 when the condition is true and 0 otherwise. This score equals 1 if the predictions are correctly ordered, and 0 if incorrectly ordered. When the predicted risk scores are exactly equal for a comparable pair (i.e., $\hat{\eta}_{i'} = \hat{\eta}_i$), the model does not provide a meaningful ordering between the two observations. To fairly account for this uncertainty, a value of $\frac{1}{2}$ is assigned. The sum of concordance scores across all comparable pairs gives us:

$$\text{Concordant pairs} = \sum_{i,i':Y_i>Y_{i'}} \left[I(\hat{\eta}_{i'} > \hat{\eta}_i) + \frac{1}{2} I(\hat{\eta}_{i'} = \hat{\eta}_i) \right] \delta_{i'} \tag{12}$$

Combining these components, the complete formula for the C-index is:

$$C = \frac{\sum_{i,i':Y_i>Y_{i'}} \left[I(\hat{\eta}_{i'} > \hat{\eta}_i) + \frac{1}{2} I(\hat{\eta}_{i'} = \hat{\eta}_i) \right] \delta_{i'}}{\sum_{i,i':Y_i>Y_{i'}} \delta_{i'}} \tag{13}$$

Advantages and Limitations of C-index The concordance index has several advantages which makes it suited for survival analysis. Firstly, the C-index is robust to censoring, as it only considers comparable pairs, (i.e. pairs where the ordering can be definetely determined). Secondly, it is quite intuitive and interpretable, as the probability that the model correctly ranks pairs according to their survival times.

However, the C-index also has limitations. The metric only evaluates ranking performance and does not assess calibration. This means that even though a model is able to rank buildings correctly, its predictions might be far away from the true observed lifetimes. Following up on this, the C-index can be insensitive to improvements in prediction accuracy, particularly when the C-index is allready high. In other words, if a model allready has a C-index of 0.8, but we get access to another feature with high predictive power, the C-index might only increase a little bit, even though the model becomes a lot more accurate at predicting lifetimes. This is because the new feature might only help in predicting features which are allready ranked correctly.

Alternative metrics suited for survival also exists. The Brier score and Integrated Brier score evaluate calibration by measuring prediction accuracy at specific time points. The approaches could be valuable for future research. We have landed on

C-index as our primary metric due to its interpretability and widespread acceptance for comparing survival model performance.

3.1.2 Feature Importance

One of our goals is to understand how different features influence demolition rates, as this could help inform policy makers. While we will not conduct an extensive comparative analysis of feature importance across models, we will examine what each individual model reveals about key characteristics.

Feature importance is calculated differently for each model type, with specific methods detailed in the respective models section. The interpretability of the results also varies between models.

It is crucial to interpret the results from feature evaluation with caution. The importance of features returned by the models will be heavily effected by the left truncation in our dataset. Left truncation results in a systematic exclusion of buildings that were demolished before our observation period. This phenomenon is discussed in more detail in Section 4.3.2, but simply put it results in features typical for buildings built a long time ago seeming more important for long survival time then they are in reality are. In addition, the feature importance measures reflect correlation rather than causation, and may be infleucned by confounding variables not present in our dataset.

We will return to these concerns when examining results for specific models, as the bias often becomes quite clear in the patterns we observe.

3.1.3 Cross-Validation

Cross-validation (CV) (Stone, 1974) is a technique in machine learning that provides a robust method for assessing how well a model's performance will generalize to an independent dataset. Instead of randomly splitting our data into training and testing sets once, CV uses a more systematic approach. Specifically, it works by repeatedly partitioning the data. In each iteration, a different portion is used for testing, while the remaining data is used for training the model. This repeated evaluation helps getting a better, more generalized estimate of the model's performance and reduces the risk of overfitting - a case where the model learns the training data, including its noise, too well and struggles with new, unseen data.

In this thesis we use *Stratified K-fold* cross-validation. This is a variation of the classic K-fold CV that returns stratified folds: each set contains approximately the same percentage of samples of each target class as the complete set. This is important for imbalanced datasets (Szeghalmy and Fazekas, 2023). In survival analysis, the event indicator ($\delta_i = 1$ for demolitions, $\delta_i = 0$ for censored) serves as our target class.

If a fold does not have sufficient demolition events, the model cannot properly learn the hazard patterns during training. For instance, a fold with only censored observations would provide no information about actual demolition times, and vice versa. Without this stratification, our nested cross-validation procedure could fail entirely in folds with little to no demolition events, making model comparison and hyperparameter tuning unreliable. A pseudo-implementation of this approach is shown in Algorithm 1.

Algorithm 1: Stratified K-Fold Cross-Validation

Input: Features X, outcomes $y = (T, E)$, folds K, model class $\mathcal{M}$
Output: Mean validation score $\overline{S}$
Separate indices where $E = 1$ (events) and $E = 0$ (censored);
Shuffle each index list independently;
for *each list (events, censored)* **do**
    for $j \leftarrow 1$ **to** *size of list* **do**
        Assign index j to fold $((j \bmod K) + 1)$;
Initialize empty list `scores`;
for $k \leftarrow 1$ **to** K **do**
    test_idx $\leftarrow$ indices in fold k;
    train_idx $\leftarrow$ all other indices;
    model $\leftarrow \mathcal{M}()$;
    model.fit(X[train_idx], y[train_idx]);
    $s \leftarrow$ model.evaluate(X[test_idx], y[test_idx]);
    `append(scores, s)`;
$\overline{S} \leftarrow \frac{1}{K} \sum_{s \in \text{scores}} s$;
return $\overline{S}$

For a complete example on how the assignment to folds is built, see Appendix A.1.

While stratified K-fold effectively figures out model performance, it doesn't give a truly unbiased estimate of a tuned model's generalization to new data, partially because hyperparameter tuning uses the entire training set. Thus, for the purposes of model training in this thesis, further explained in Section 5, Section 6, and Section 7, we use nested cross-validation (NCV). NCV extends the CV framework by implementing two loops: an outer loop to evaluate model performance, and an inner loop dedicated to hyperparameter tuning. As shown by Varma and Simon (2006), this method is particularly useful because it helps obtain honest scores for model performance and, at the same time, determines the best settings for the model hyperparameters. We can think of it as having two rounds of this data partitioning

idea.

The outer loop first splits our entire dataset into K folds. Then, it goes through each of these folds one by one. In each turn, $K-1$ of these folds combine to become the training ground, while the remaining single fold becomes the unseen test set we will use to evaluate model performance. Within each outer loop iteration, the training set undergoes its own cross-validation process (the inner loop) to determine the optimal hyperparameters. Once these optimal hyperparameters are identified, they are used to train a model on the entire training set from the outer loop, which is then evaluated on the unseen test fold. For a visualization of how the nested stratified cross-validation works, see Appendix A.2.

One might ask: why go through all this? It seems computationally expensive, and adds a lot of complexity to the problem. These are valid concerns. However, the advantages it offers in terms of robust model evaluation and prevention of data leakage are often well worth the extra effort. By making sure hyperparameters are tuned without any influence from the outer test set, we prevent some subtle data leakage. Furthermore, NCV gives a more honest estimate of model generalization. As discussed by G. C. Cawley and Talbot (2010) standard CV can lead to selection bias when used for both hyperparameter tuning and performance estimation. These considerations justify the additional computational cost. On the other hand, an interesting approach, flat cross-validation, co-authored by the same author arguing for NCV, Wainer and G. C. Cawley (2021), shows that good results can be found at lower computational expenses - something definitely worth exploring in the future.

3.1.4 Model Comparison

After training and testing the different models we will analyze the differences between the models performances. To determine whether the differences in model performance are statistically significant, we conduct pairwise comparisons using the correlated t-test proposed by Nadeau and Bengio (2003). This test accounts for the correlation between folds in K-fold cross-validation and adjusts the standard error accordingly.

Let r_i denote the performance difference between two models on outer fold i, and let $\bar{r}$ and s denote the mean and sample standard deviation of these differences. The corrected t-statistic is computed as:

$$t_{stat} = \frac{\bar{r}}{s \cdot \sqrt{\frac{1}{J} + \frac{1}{K(1-\frac{1}{K})}}}, \tag{14}$$

where K is the number of folds used in cross-validation, and J is the number of observed performance differences r_i used in the test statistic. In our case, we use

one performance difference per outer fold, so $J = K = 5$. The resulting t-statistic is compared against the Student's t-distribution with $J - 1$ degrees of freedom.

Standard paired t-tests assume independence between test folds, which does not hold in K-fold cross-validation due to overlapping training sets. The correlated t-test adjusts the standard error to account for this dependency, leading to more reliable inferences about model performance differences. This correction is especially important when the number of folds is small or when nested cross-validation is used, as it helps prevent underestimation of variance and inflated Type I error rates (Nadeau and Bengio, 2003) and (Herlau, Schmidt, and Mørup, 2023).

3.2 Technical Implementation

Having established our framework for evaluating different survival analysis methods, we now describe how we actually implement these approaches in practice. Moving from theory to working code brings its own challenges, especially when dealing with a dataset as large as ours.

Our implementation has significant computational challenges. Ideally, we would train and evaluate all survival methods on the complete dataset consisting of roughly 4.4 million observations. However, this scale of analysis quickly becomes computationally expensive on our standard laptops, particularly for the more complex ensemble and deep learning methods that require hyperparameter tuning through nested cross-validation.

These computing limits are an important part of our proof-of-concept design. While survival analysis methods should be able to handle datasets this large, doing proper model comparison and tuning would need high-performance computing (HPC) resources that we have decided not to pursue due to practical limitations. This limitation shows why our work serves as a proof-of-concept rather than a ready-to-use prediction tool.

This section explains our computing setup, which software libraries we chose, and the key decisions we made to ensure our results are reproducible and our model comparisons are fair.

Data Loading and Sampling Given the large amounts of data, we implement a `DataLoader` class that can handle different ways of loading the data. It is built with a sampling functionality that allows us to extract subsets of the full dataset either as a percentage of the total data (for example, 0.1% for this proof-of-concept thesis) or an absolute number of rows. However, rather than using simple random sampling, we implement a stratified sampling to ensure that smaller datasets maintain the same proportion of demolished versus non-demolished buildings as the original dataset. It is important that we are aware of which strategy we use for sampling – for example,

random sampling could accidentally create samples with too few or too many events, making model training difficult or unreliable.

To speed up repeated experiments, the data loader includes a caching system that saves processed datasets to disk. Once a sampled dataset is created and cached, later runs can load it directly without repeating the sampling process. This is particularly nice when working with the dataset over a longer period of time and want to train different models on the exact same data sample for fair comparison.

Architecture We use `Python` for all our analysis, taking advantage of its robust ecosystem for survival analysis and machine learning. Our workflow relies heavily on standard libraries such as `NumPy`, `Polars`, and `Pandas`, while also incorporating specialized packages like `Lifelines`, `sksurv`, `pycox`, and `torchtuples` for survival modeling.

Our implementation follows a modular design that separates concerns and enables fair comparison across all survival analysis methods. The architecture consists of four main components that work together to handle the complete modeling pipeline.

The model directory contains individual class implementations for each survival analysis method we test. Each model class follows the same structure, with standardized methods for fitting, prediction, and evaluation. This consistent structure makes us able to compare traditional statistical methods like Cox Proportional Hazards directly with ensemble methods like Random Survival Forests and deep learning approaches like DeepSurv, all using identical evaluation procedures. The configs directory stores hyperparameter specifications for each model type in separate configuration files.

We implement a cross validation framework for the stratified nested cross-validation approach. The framework handles the challenge of maintaining balanced event rates across folds while performing both hyperparameter tuning in the inner loop and performance evaluation in the outer loop.

The training pipeline is orchestrated by a central script that coordinates the entire modeling process. This script loads the data, initializes each model type with its respective configuration, and runs the nested cross-validation procedure. It is important that we centralize this process, because it allows us to receive identical data splits for each survival analysis method.

Practical Model Training A major limitation of our thesis is that we only can work with **0.1%** of the full BBR dataset, when running the full evaluation setup on all 6 models. This is due to computer limitations when using our personal laptops for this analysis. The nested cross-validation framework needs to train thousands of models. Each model training involves fitting the model, making predictions, and

calculating performance scores. When we multiply this across the full framework, these operations need more memory and processing power than regular computers can handle.

We decided not to use our university's High Performance Computing (HPC) resources for this thesis. While HPC systems would let us work with more data or even the complete dataset, learning to use these systems would take significant time and effort that goes beyond what we can do for this proof-of-concept study. Our goal is to show how these methods work and compare how well they perform, which we hope can be achieved reasonably well with even a very small sample of the data.

With that limitation in mind, our nested cross-validation framework consists of four main algorithmic components that work together to provide an estimate of the true model evaluation and comparison. We begin with the overall training pipeline that coordinates the entire process.

Algorithm 2: Main Training Pipeline

Input: Dataset path, model types
Output: Performance results $\mathcal{P}$
for *each model type* **do**
    Load hyperparameter grid from config file;
    Initialize model class;
Load dataset with sampling ;
Preprocess features and survival outcomes;
$\mathcal{P} \leftarrow$ NestedCrossValidation(features, outcomes, models);
Save results;
return $\mathcal{P}$

First the pipeline loads the hyperparameter grid for each model and initializes the class. It then loads the dataset, and prepares it for model training.

Once data preparation is complete, the pipeline calls the nested cross-validation procedure, which forms the core of our evaluation framework. While we usually only work with a subset of the full dataset, the nested structure is in theory essential for survival analysis because it provides a more unbiased performance estimate while accounting for hyperparameter optimization. We use a setup with 5 outer and 5 inner folds, because it is commonly used, for instance, by Wainer and G. Cawley (2018).

Algorithm 3 shows that for each outer fold, the algorithm reserves a test set that remains completely unseen during hyperparameter tuning. The remaining training data is then used for parameter optimization through the inner cross-validation loop. After finding optimal parameters, a final model is trained on the complete training set and evaluated on the held-out test data. This procedure makes sure that performance

estimates reflect something closer to generalization ability rather than overfitting to specific data splits.

Algorithm 3: Nested Cross-Validation

```
Input: Features X, survival outcomes y, model configurations
Output: Model performance results
Create stratified folds using survival event indicators;
Initialize results storage;
for each outer fold do
    Split data into training and test sets;
    for each model type do
        optimal_params ← HyperparameterTuning(training_data);
        Train model with optimal_params on full training set;
        Evaluate model on held-out test set;
        Store performance metrics and parameters;
Aggregate results across all folds;
return results
```

Algorithm 4: Hyperparameter Tuning

```
Input: Training data, model class, parameter grid
Output: Optimal parameters θ*
Create inner stratified folds from training data;
Initialize best_score ← 0;
for each parameter combination θ in grid do
    Initialize validation scores list;
    for each inner fold do
        Split training data into fit and validation sets;
        Train model with parameters θ;
        Evaluate on validation set;
        Record validation score;
    avg_score ← average of validation scores;
    if avg_score > best_score then
        best_score ← avg_score;
        θ* ← θ;
return θ*
```

The hyperparameter tuning algorithm performs a full grid search within each outer fold's training data. It creates additional stratified folds to ensure that parameter se-

lection is based on robust performance estimates rather than some train-validation split that could just be a fluke, even if it has great performance. For each parameter combination, the algorithm trains models across multiple inner folds and averages their validation scores. We choose full grid search over more sophisticated hyperparameter optimization methods because our parameter grids are very small, which makes it doable. While we could have expanded the grid search and used methods such as random search or Bayesian optimization, we went for the simpler approach given the proof-of-concept nature of this thesis.

To illustrate the computational demands of our nested cross-validation framework, we can calculate the total number of models that need to be trained. Consider our setup with 5 outer folds, 5 inner folds, and 5 different survival analysis methods.

Let x_i represent the number of hyperparameter combinations for survival analysis method i, where $i \in 1, 2, 3, 4, 5$. For each survival analysis method i, the nested cross-validation framework requires training $x_i \times 5 = 5x_i$ models during hyperparameter tuning across the 5 inner folds, plus 1 additional model for final evaluation on the outer test set. This gives us a subtotal of $5x_i + 1$ models per method per outer fold.

When considering all model types together within a single outer fold, we sum across all methods to get

$$\sum_{i=1}^{5}(5x_i + 1) = 5\sum_{i=1}^{5} x_i + 5 \text{ models.} \tag{15}$$

Since our framework uses 5 outer folds, the total number of models trained across the entire nested cross-validation procedure becomes

$$5 \times \left(5\sum_{i=1}^{5} x_i + 5\right) = 25\sum_{i=1}^{5} x_i + 25 \text{ models.} \tag{16}$$

This can be expressed more generally as

$$\text{Total Models} = K_{outer} \times \left(\sum_{i=1}^{M} K_{inner} \times x_i + M\right), \tag{17}$$

where $K_{outer} = 5$ outer folds, $M = 5$ model types, x_i is the number of hyperparameter combinations for method i, and $K_{inner} = 5$ inner folds. We will get into the specifics of the amount of combinations for each method later, but for the sake of the argument, assume the average number of combinations is 100 per survival analysis method. This gives us

$$25 \times 500 + 25 = 12500 + 25 = 12525 \text{ models.} \tag{18}$$

This calculation demonstrates part of the reason why we can only work with 0.1% of our full dataset. Training over 12,500 models, even on a subset of the data, requires a lot of computational resources and time, once again highlighting why this approach serves as a proof-of-concept rather than a production-ready system.

4 Data and Preprocessing

In the following sections we describe the data we use in this thesis, as well as the methods used for cleaning it. We begin with a section on the acquisition and cleaning of the data, presented alongside relevant considerations. We then apply descriptive statistics and data visualizations to understand the variables in the dataset, and the interactions between them. Finally, we present an assessment of the quality of the data, both in terms of bias and accuracy.

Since the dataset originates from Denmark, all feature names are in Danish. A translation of all relevant features is provided in Appendix A.3.

4.1 Data Acquisition and Cleaning

This section describes the data sources used in the project and the methodology for data cleaning and preparation. It also covers key considerations and decisions made throughout this process.

4.1.1 Data Sources

The Danish Building and Housing Register (BBR) is the primary data source for this project (Bygnings- og Boligregistret (BBR), n.d.[a]). Established in 1977, BBR is a public registry containing information on properties, buildings, and construction activities throughout Denmark. It serves purposes such as property valuation, urban planning, and statistical reporting. Data acquisition from BBR was done through the "Datafordeler" web service, specifically through its HTTPS file download service, which provided the data in JSON format (Klimadatastyrelsen, n.d.).

Within BBR, the analysis focuses on the "Bygninger" dataset, which contains *complete* building registrations from 2017 onward. Each entry in the dataset includes the registration reason, which is identifiable through the `Business Process` feature. A significant system update for BBR happned in 2017 (Skatteministeriet, 2019). The extracted dataset contains complete registrations from after this update, however, many historical entries were also converted to the new system. We note that the completeness of data for registrations prior to 2017 cannot be guaranteed.

A supplementary data source was provided by Rune Andersen, an Assistant Professor at the Department of Environmental and Resource Engineering at The Technical University of Denmark (DTU). This dataset, which contains demolition records from 2010 to 2020, has already been through initial cleaning, and its methodology is detailed in Andersen and Negendahl (2023). However, some additional preprocessing is done before merging it with the primary BBR data. The overlap period between Andersen's dataset and the BBR data is at least four years (2017 - 2020). In cases

of overlapping information, data from the BBR will be prioritized because it is an actively updated source.

4.1.2 Data Cleaning Methodology

Before merging the two datasets, we perform initial cleaning on them separately. The core of the cleaning is done with the Polars Python library. This open-source tool, built on Rust, is recognized for its blazingly fast performance, and is thereby well suited for handling large datasets such as BBR. Figure 2 illustrates the full data cleaning pipeline, and Table 2 shows how many entries were removed at each step of the process.

Cleaning of the BBR data The BBR "Bygninger" dataset, initially obtained in JSON format, is converted to CSV for more efficient processing, since it doesn't include metadata or nested structures. The initial cleaning involves removing administrative features such as `ID Namespace` and `Quality of Coordinate Set`, as they are not relevant for analyzing building survival. The feature `Source` is then added and given a value of 1 for all entries, to indicate that they come from BBR.

The current status of a building can be found in the feature `Status`, where code 10 corresponds to the building being historic. We use this code to indicate that a building has been demolished.[§] Thus, the process for identifying demolished buildings and assigning a demolition year is as follows:

- A building is considered demolished if its `Status` is 10.
- If a building record initially shows a `Status` = 10 but later has an entry with a different `Status`, it is removed. Logically, a building cannot reappear after being marked as historic/demolished.
- A new feature, `Demolition Year`, is created to store the year of demolition.
- For all buildings identified with a `Status` = 10, their demolition year is extracted from the `Effect From` feature of the first entry where `Status` is 10. This accounts for administrative updates that might occur after a building's demolition.

Certain feature codes within the BBR dataset have been discontinued. When calculating the demolition rates for entries with these codes, they are found to be unusually high. It was also noticed that the construction years for these old-code entries were similar to those with current codes. This suggests a problem with how this data is

[§]Read more about how we choose the demolition indicator in Section 4.1.3

handled: entries with old codes might have been marked as demolished, and new entries created with updated codes. Because of this, we remove all entries that use discontinued codes. We will discuss this decision and its impact in more detail in Section 4.1.3.

Once the `Demolition Year` column has been added and the erroneous demolition entries are removed, we make sure only one record remains per building. We assume the latest entry provides the most current and accurate information, so for each building, we keep only the entry with the most recent `Effect From` value.

The use case from the feature `Use Category` is then grouped into the categories: Transport, Institution, Production, Commerce, Leisure, Agriculture, Housing, and Other. The feature `Municipality Code` is also mapped into their corresponding `Region` (Danske Regioner, n.d.).

The BBR dataset is now pre-cleaned and ready for merging with Andersen's dataset, which will also undergo some cleaning.

Cleaning of Andersen's data The cleaning process begins by removing entries lacking a value in the `Effect From` feature, as this is essential for determining the demolition date. Buildings with multiple entries are dropped, as we do not know which entry to keep in this case. Only a very small fraction of buildings (0.2% of the entries that did not have null in `Effect From`) have duplicate entries, and we opted to remove these rather than make assumptions of which to keep. The feature `Source` is then added, and now given a value of 2 for all entries, to indicate that they come from Andersen's dataset.

For Andersen's dataset, all entries are known to represent demolished buildings. Therefore, when adding the `Demolition Year` feature, each entry is directly assigned the year from its corresponding `Effect From` value.

Similar to the BBR data, entries containing outdated feature codes are removed due to their unusually high recorded demolition rates. Finally, use cases within Andersen's dataset are grouped into the same `Use Category` groups established for the BBR data, and `Municipality Code` values are mapped to their respective regions.

Merging the datasets After individually cleaning both the BBR and Andersen datasets, we merge them into one joint dataset. The merging process involves appending all entries from Andersen's dataset to the BBR dataset, keeping all features from both sources. Features unique to one dataset will have empty values for entries coming from the other, thus ensuring the full preservation of all records. Following this initial merge, any duplicate entries appearing in both datasets are identified and removed, with preference given to the entry sourced from BBR.

The feature `Year of Construction` is very important for the intended analysis

of the dataset. Therefore, all entries that do not have a value for this feature are removed. While definitively pinpointing the first building in Denmark is challenging, we use the construction of Korsbrødregården around 1425-1430, as stated by Historie Online (n.d.), as a pragmatic lower bound for possible construction years. It's also notable that a substantial number of entries incorrectly list a construction year of 1000 (and even year 0, as if from a biblical era). Instead of attempting to impute these values, we remove all entries with a construction year recorded before 1426 or in the future. This approach deals with these clear errors without making assumptions that might be wrong.

Having completed the individual cleaning processes, the two data sources are now merged into a single dataset, with all entries considered valid for analysis. Further preparation, such as one-hot encoding and feature extraction, is required before model training. These steps are detailed in the model training sections.

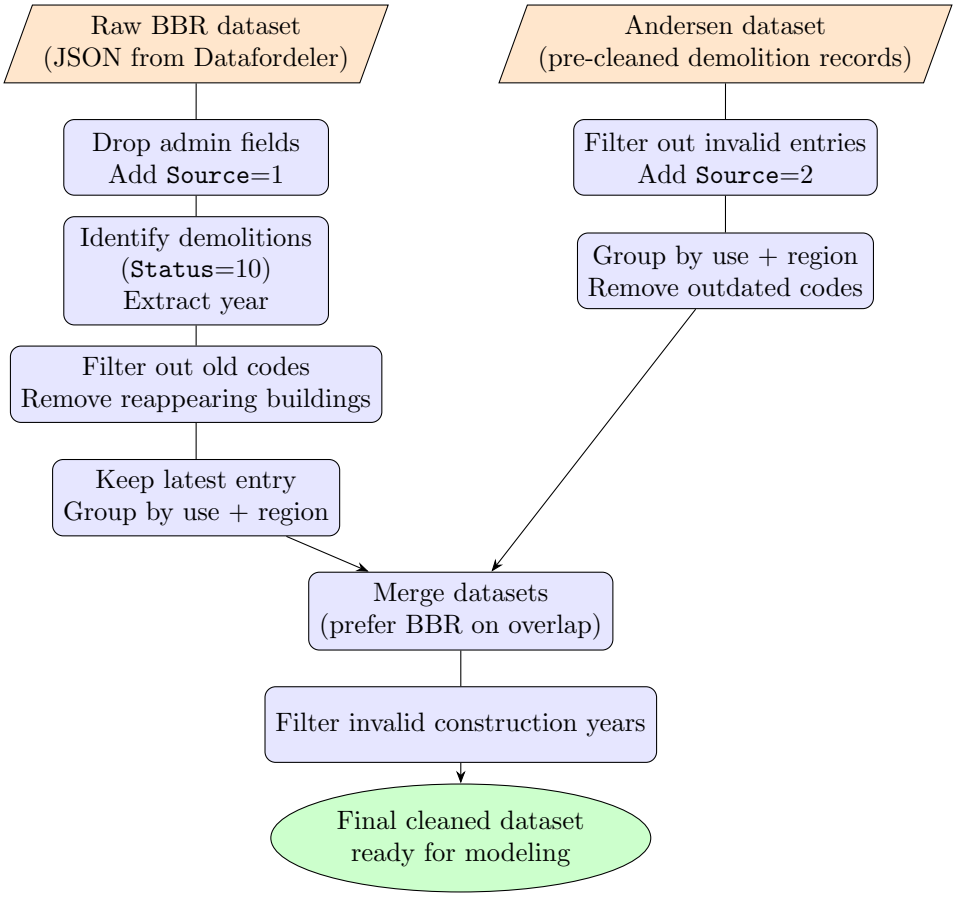

Figure 2: Pseudo flowchart illustrating the data cleaning and merging process for the BBR and Andersen datasets.

Step	Before	Removed
Cleaning BBR		
Converting to CSV	6 087 324	0 (0.0%)
Remove false demolitions	6 087 324	5 338 (0.1%)
Remove discontinued codes	6 081 986	1 217 507 (20.0%)
Keep latest entry	4 864 479	0 (0.0%)
Cleaning Andersens		
Import file	149 013	0 (0.0%)
Remove null `Effect From`	149 013	25 115 (16.9%)
Remove duplicates	123 898	188 (0.2%)
Remove discontinued codes	123 710	47 036 (38.0%)
Cleaning Merged		
Merge datasets	4 865 162	0 (0.0%)
Remove duplicates	4 865 162	0 (0.0%)
Remove faulty construction years	4 865 162	428 884 (8.8%)

Table 2: Overview of number of unique ID's before and number of unqiue ID's removed at key cleaning steps. Removed is a percentage of the "step count", not of the total number of buildings.

4.1.3 Determining the Demolition Indicator

Determining how to indicate whether a building has been demolished is surprisingly complicated. It would be advantagous to have a feature indicating if a building has been demolished, however this is not the case for the BBR dataset. Throughout the project we have gone back and forth between several different methods of determining demolitions. The following sections show our iterative process of exploring and refining various methods to define a good and reliable demolition indicator.

Initial Approaches to Identifying Demolitions Initially, we identified demolitions using the `Business Process` feature, where a code of 3 is intended to mark an update due to demolition (Bygnings- og Boligregistret (BBR), n.d.[b]). This method is consistent with the thesis by Droob and Nybroe (2024), and seems intuitive. However, uncertainty about its accuracy made us consult BBR directly. While they couldn't provide a precise instruction, BBR suggested we look into the `Status` fea-

ture, where a code of 10 indicates that the building is 'historic'. We therefore changed our demolition indicator to be `Status` = 10.

However, after implementing the `Status` = 10 indicator, we observed unexpected behavior within our data. Demolition rates for specific features were found to be unrealistically high, often above 90%, which is significantly above what we observe for other features ($1 - 15\%$). Such inflated rates caused these features to be near-perfect indicators for demolition. By far, these features then became the most important feature for our models. Upon further investigation, we found that these problematic codes were marked as being discontinued, and had often been split up into several more granular categories. Demolition rates for all discontinued codes can be found in Section A.5.

For instance, the feature `Use Category` previously used code 210 for buildings used in commercial production related to agriculture, mineral extraction or similar. This specific code was discontinued and replaced by nine new, more specific categories, such as a new code 211, which is a code designated for pig stables. Importantly, these new categories have much lower demolition rates, aligning with our expectations ($1 - 15\%$). The demolition rates for these categories can be seen in Table 3.

Code	**Description**	**Demolition Rate**
210	(Discontinued) Building related to agriculture, horticulture, mineral extraction, or similar	92.7%
211	Pig stable	6.4%
212	Cattle, sheep, etc. stable	5.0%
213	Poultry house	11.5%
214	Mink shed	9.0%
215	Greenhouse	18.3%
216	Barn for feed, crops, etc.	5.6%
217	Machine shed, garage, etc.	4.4%
218	Barn for straw, hay, etc.	5.3%
219	Other building for agriculture, etc.	9.6%

Table 3: Demolition rates for the discontinued code 210 and the newer, more specific agricultural use codes 211–219. The significantly lower rates for the newer codes support the hypothesis of re-registration rather than actual demolition.

We could also see that the construction years in these new categories had similar distributions to the construction years in the old category. We therefore hypothesize

that during the creation of these new categories, buildings previously associated with the old codes were assigned *new IDs*, while their original IDs were marked as historic (`Status` = 10).

Our theory is that code 210 (or similar discontinued codes), may have previously represented multiple buildings grouped together, for instance, a pig stable, a cow stable, and a chicken coop could all be grouped under one ID. We believe these groupings were then split into individual IDs, and given more specific codes. We believe this re-registration process is the underlying cause of the artificially high demolition rates in the discontinued categories.

In Figure 3 the number of buildings constructed per year for both the old code 210 and the new codes $211 - 219$ can be seen. The plot shows that the total number of buildings in the new categories is much higher than in the old categories, that the ratio between the two groups remains fairly constant over time and that both span a similar range of construction years, all three points supporting our hypothesis.

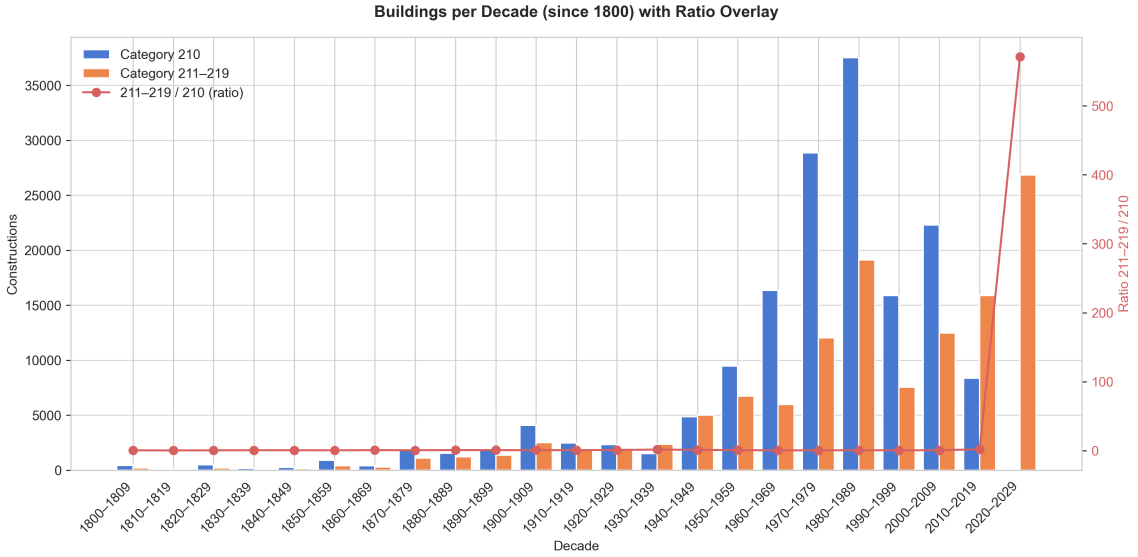

Figure 3: Number of buildings constructed per year for code 210 (discontinued) and the new codes 211–219. The red line indicates the ratio between the two groups.

One benefit with using `Status` = 10 is that nearly all demolitions from Andersen's supplementary dataset were also correctly identified within the BBR data. This reduces the reliance on the supplementary dataset, creating a more robust primary data source – the positive here: `Status` = 10 seems like a more complete identifier, and we would not *have* to use Andersen's data.

Refining the Demolition Indicator After discovering that certain categories gave us excessively high demolition rates when using `Status` = 10, we thought of an alternative approach for identifying demolitions. The next approach we tested was looking at the buildings that satisfied *both* `Status` = 10 *and* `Business Process` = 3. The idea here is that adding the `Business Process` requirement would filter out the buildings that were marked as historic for different reasons than being actual demolitions.

However, our analysis showed us that the number of demolitions identified using this combined method was very similar to our original approach that relied only on `Business Process`. Furthermore, we discovered a significant drawback: buildings from Andersen's supplementary dataset were to a much larger extend included in the data after merging. Since we prioritize BBR data over Andersen's data when duplicates exist after merging, this told us that a substantial number of buildings that should be marked as demolished within the BBR dataset were not being captured by this new method.

Given these findings, we returned to using `Status` = 10 to mark demolitions. However, we introduced an important refinement: we removed all IDs associated with a *discontinued* feature. As mentioned above, our assumption is that these buildings have been re-registered under new entries with updated categories. It should be noted that there could be other reasons for a building to be marked historic that we haven't considered. This means there's a risk we might be overcounting demolitions with this method. Nevertheless, we are willing to accept this risk considering that using the other methods, we are certain we would be undercounting.

Thus, our final method for determining demolitions is to identify entries where `Status` is 10, and then remove any entries with discontinued features. While not a perfect solution, it's the most effective method we've found. The full overview of the iterative process can be seen in Figure 4.

4.2 Exploratory Data Analysis

In this section, we explore the dataset's key characteristics by computing summary statistics and plotting survival curves for the most important features. The insight gained in this section will be valuable for both data quality assessment in Section 4.3 and for evaluating the results from the models trained in later sections.

4.2.1 Initial Data Overview and Summary Statistics

Data Dimensions We begin our exploratory analysis by formally describing the shape and basic properties of the dataset. Let $\mathcal{D} \in \mathbb{R}^{n \times p}$ denote the full dataset,

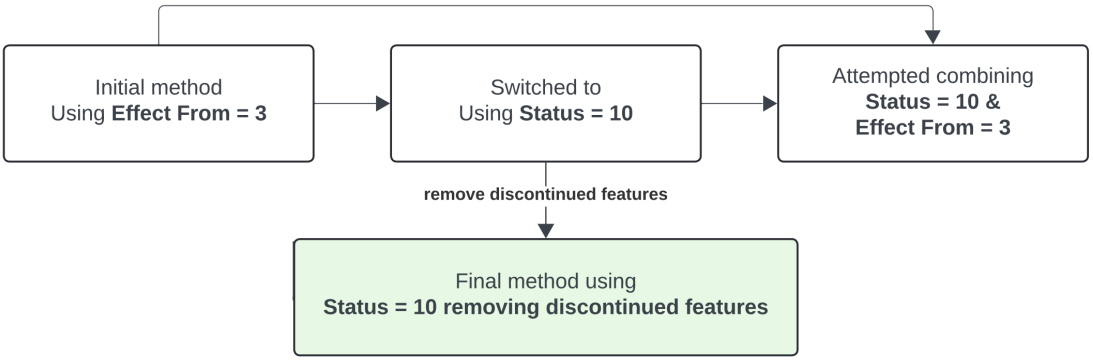

Figure 4: Iterative process for finding the demolition indicator. The final method uses the 'historic' code, and specifically removes the problematic building entries to get a more accurate count.

where:

- $n = 4,436,278$ is the number of building entries
- $p = 85$ is the number of features

This dataset includes both continuous and categorical variables, many of which exhibit a high degree of missing values (see Appendix A.11). This version of the dataset is not yet ready for use in machine learning models. Prior to modeling, we perform additional preprocessing steps such as one-hot encoding of categorical variables, imputation of missing values (when possible), and construction of new features. These transformations are discussed in Section 4.4.

However, the dataset presented here serves as a clean, structured representation of the BBR data at the individual building level. It is particularly well suited for researchers interested in analysis of static building properties and their lifespans rather than time-evolving building states.

A large fraction of the buildings are still standing and thus do not have an observed demolition year, resulting in a high proportion of right-censored data points. Table 4 presents descriptive statistics for two core variables: the construction year and the demolition year.

The table shows that the vast majority of buildings are not labeled with a demolition year, confirming the heavily censored nature of the dataset (as detailed in

Statistic	Demolition Year	Construction Year
Observations	253521	4436278
Missing Values	4182757	0
Mean	2017.766	1968.981
Standard Deviation	5.570	42.279
Minimum	2000	1426
First Quartile	2015	1953
Median	2019	1976
Third Quartile	2022	2000
Maximum	2025	2025

Table 4: Summary statistics for demolition and construction years. Demolition year data has many missing values, because most buildings are still standing.

Section 2.1.3). The construction years span nearly six centuries, though most buildings were constructed after World War II. These patterns will inform the structure of survival models introduced in later sections.

4.2.2 Visual Data Exploration

To get a better understanding of the dataset, we will now explore it visually. This exploration should help us find patterns that might not be so obvious when only looking at summary statistics as well as giving us a better feel for the data we are working with.

Lifetime Trends by Region The patterns of demolition and new construction vary significantly across different regions of Denmark, helping us understand how the country's built environment is changing over time.

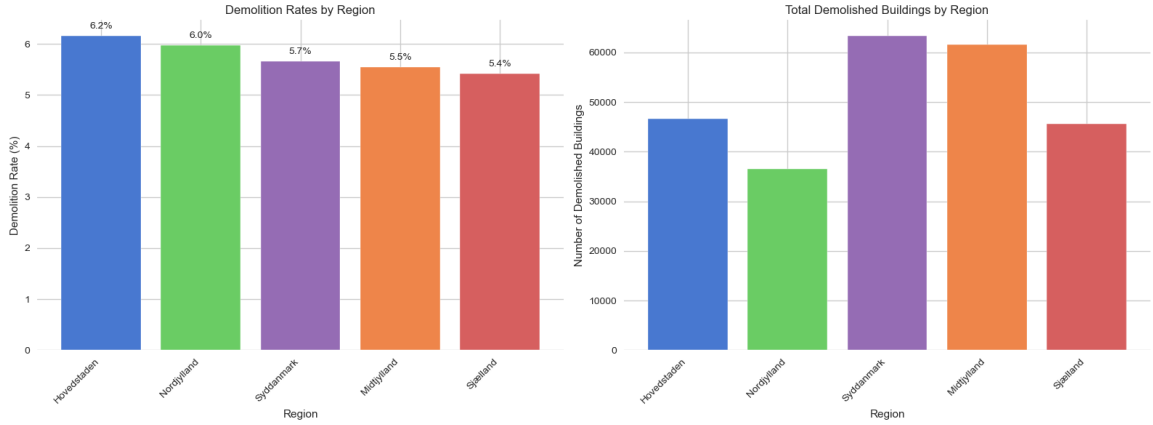

Figure 5: Regional demolition patterns showing both demolition rates and total demolished buildings by region.

As seen in Figure 5, Hovedstaden leads with the highest demolition rate at 6.2%, followed closely by Nordjylland at 6.0%. The remaining regions show progressively lower rates: Syddanmark (5.7%), Midtjylland (5.5%), and Sjælland (5.4%). This suggests Hovedstaden has the most intensive demolition rate relative to its building stock. However, the absolute numbers tell a different story. Syddanmark and Midtjylland dominate in total demolished buildings (approx 60,000 each), despite having lower demolition rates. Hovedstaden, with the highest rate, has demolished only approx 47,000 buildings. Nordjylland has the lowest absolute count (approx 37,000), while Sjælland falls in the middle (approx 46,000).

This pattern indicates that although the Capital Region (Hovedstaden) has the highest demolition rate relative to its total number of buildings, the larger regions – Southern Denmark (Syddanmark) and Central Jutland (Midtjylland) – account for more demolitions in absolute terms, simply because they have more buildings. The difference in demolition rates might also be correlated with the different distributions of building types in the regions. This can be seen in Figure 8, and we return to this later.

The stacked area chart in Figure 6 reveals Denmark's construction boom and decline patterns from 1800 to approximately 2010. The data shows minimal construction activity through the 19th century, with dramatic acceleration beginning around 1900.

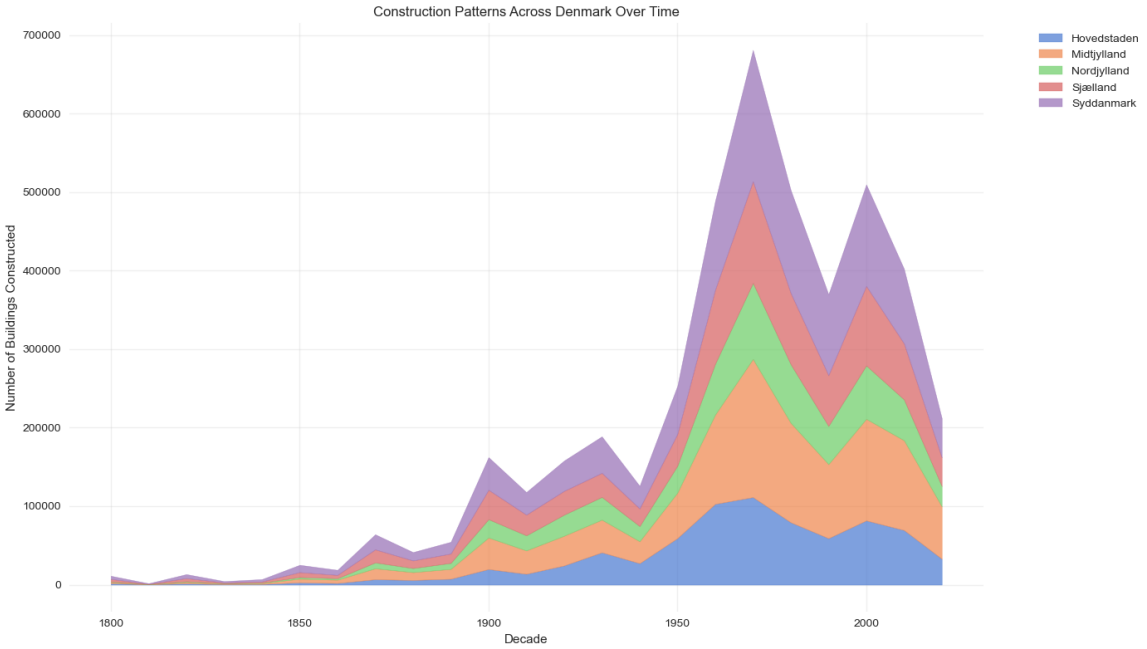

Figure 6: Regional construction pattern through time stacked on top of each other

Similar to what we saw in the histograms on construction, there is a massive construction peak after World War 2, reaching nearly 700,000 buildings cumulatively across all regions. Again, this could reflects Denmark's post-war economic growth, but also be tied to the limited time span of out dataset.

Syddanmark dominates the peak construction period, contributing the largest share during the boom years. Midtjylland and Nordjylland also have substantial contributions, while – somewhat surprisingly – Hovedstaden has a more modest but consistent levels throughout. This could be due to the fact that in BBR, a large building housing tens of families is registered as a single building – and so is a simple house anywhere else in Denmark.

After the post war peak, construction activity drops significantly but is still high compared to pre-1950 levels. A secondary, smaller peak appears around 2000. The construction then begins to decline rapidly, perhaps as discussed in Dam et al. (2011) due to the housing bubble that burst in 2007.

If we take a look at building lifetimes by region as seen in Figure 7, we see that the average building lifetimes cluster between 47.6 and 60.1 years across all five Danish regions, with most regions falling in the 57-60 year range.

We see similar lifetimes for the regions although Hovedstaden stands out with a lower mean lifetime of 47.6 years. This tells us that regional factors like climate,

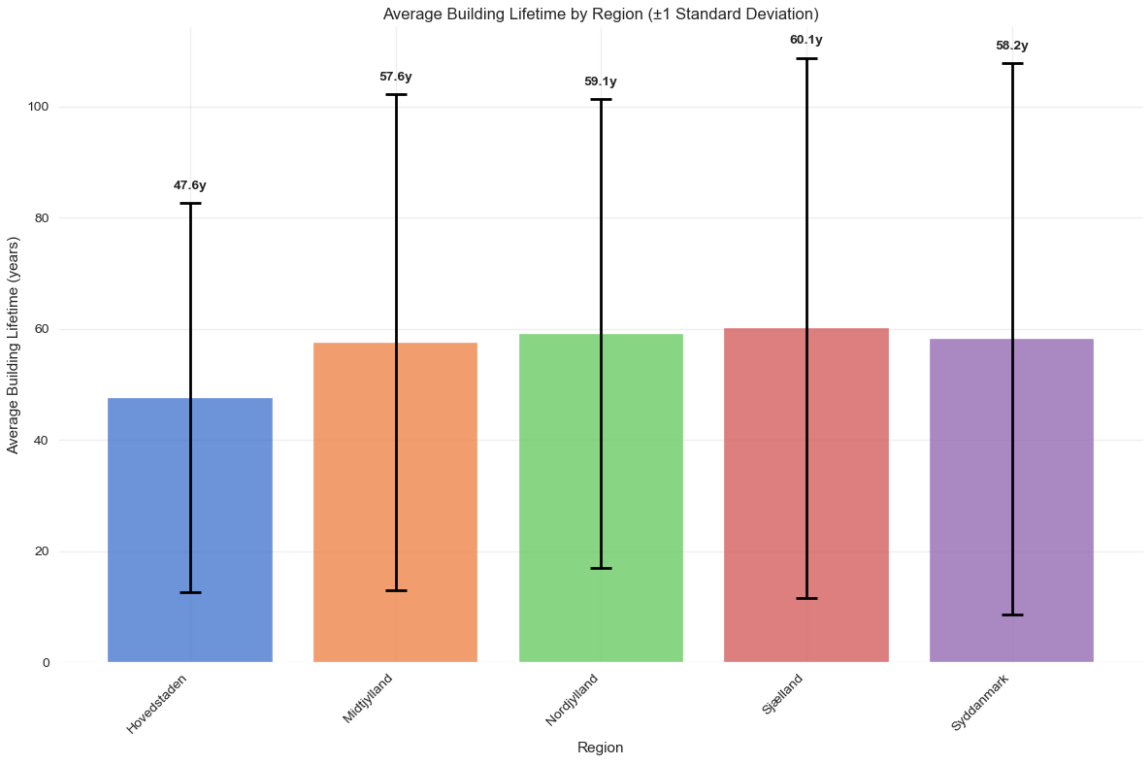

Figure 7: Bar chart displaying average building lifetime by Danish region with error bars showing one standard deviation.

development pressure, or local building practices may not have a significant impact on average lifetimes. Particularly when looking at the large error bars ($\pm 1\sigma$), region seems to be a weak predictor of lifetime. The error bars all span more than 60 years, which is much bigger than the differences between the mean lifetime of the regions, which are only 12.5 years apart for the bigegst gap (Hovedstaden and Sjælland).

This reveals a need for granular analysis. The pattern suggests that material type, construction method, building use, or other features may be more important determinants of building lifetime than regional location. A building in `Use Category` housing in Hovedstaden might have more in common with another Housing building in Nordjylland than with an agricultural building in the same region. Indeed Figure 8 shows that Hovedstaden has relatively fewer buildings in the `Use Category` Agriculture, which generally has a high mean lifetime and relatively more in the category other, which tends to have shorter lifetimes (see Table 5). Then again, `Use Category` may itself be a bad predictor with its own confounders such as materials or area - this will be investigated more in the next paragraph. For further visualization, a violin

plot describing building lifetimes by region can be seen in Appendix A.7.

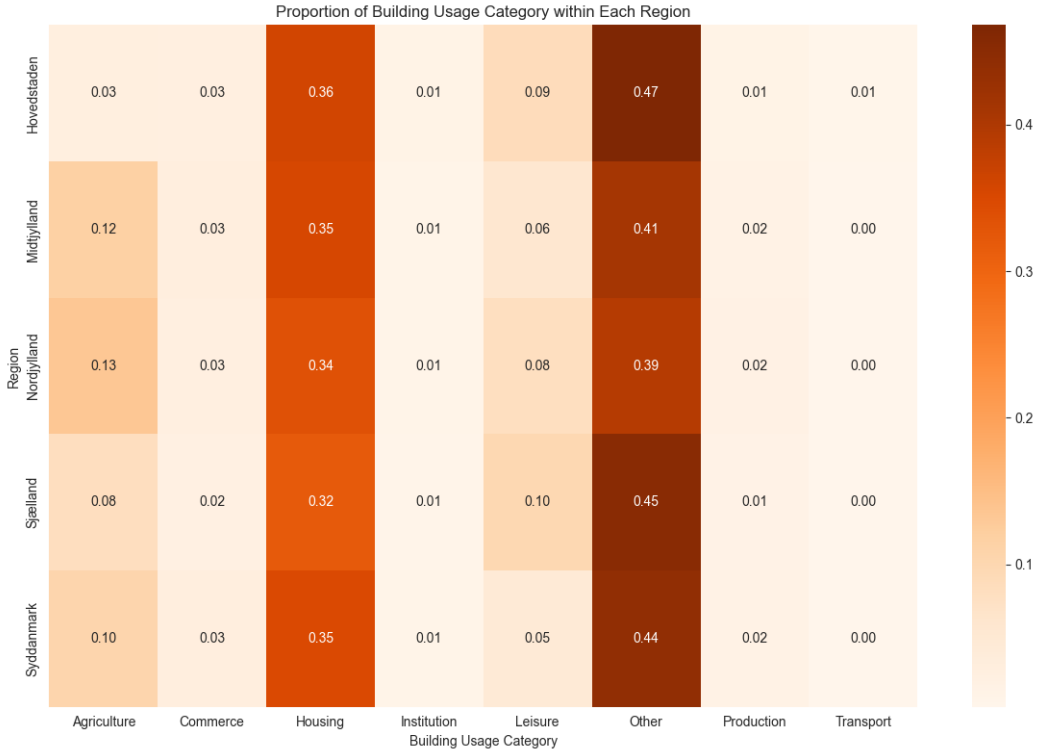

Figure 8: Heat map showing proportions of buildings in each category for every region

We get a slightly more nuanced view of the lifetime distribution, when we look at Figure 9. It's worth noting that binning continuous lifetime data into discrete categories like "Short," "Medium," and "Long" is somewhat artificial – building lifetimes exist on a spectrum, and what is actually "short" versus "medium" is inherently subjective and dependent on the context.
With this in mind, all regions show similar lifetime distributions. The "Short" category (20-50 years) dominates everywhere, having roughly 30-35% of buildings. The "Medium" category (50-80 years) forms the second-largest segment at around 20-30%. Again, this consistency supports the idea that regional location doesn't have much influence on building lifetime patterns.

Impact of Building Usage on Lifetime Trends Data on lifetimes and demolitions can be hard to understand without structure. Having established the basic structure of our dataset, we now turn our attention to visualizations. We aim to

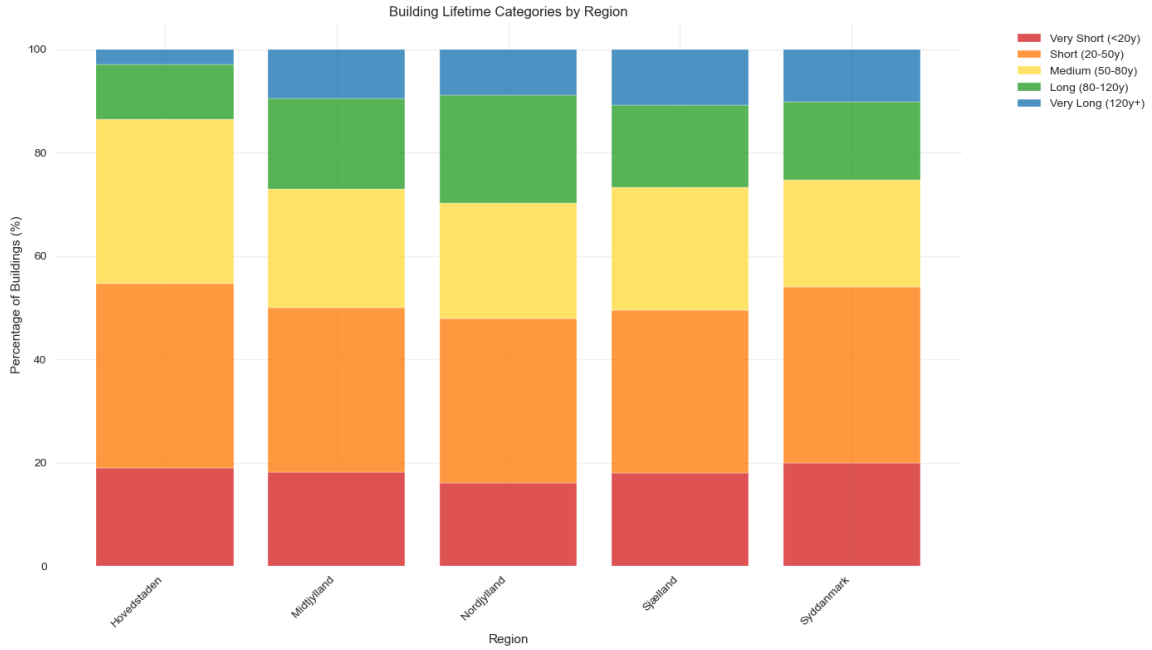

Figure 9: Stacked bar chart showing the distribution of building lifetime categories across the five Danish regions.

highlight patterns in how buildings are used and how long they last, helping us understand the data better.

Figure 10 shows the temporal distribution of building construction across different usage categories. The most striking pattern is the sharp increase in construction activity after World War II, with housing being the most frequent. This post-war construction boom might reflect Denmark's urbanization and economic growth after the war (Vestergaard and Haagerup, 2012). Agricultural buildings show more consistency throughout the timeline, while institutional and commercial buildings only appear more in recent decades.

The large "Other" category makes sense when we look at what it includes: garages, carports, sheds, greenhouses, etc. Intuitively, we can assume these smaller structures are built much more often than main buildings as properties add facilities over time. We have chosen to cut the dates at year 1800 for this visualization, since very few buildings are constructed before that. However, a full visualization can be seen in Appendix A.6.

The key takeaway from this is that most buildings in our data were built not too long ago, which matters for our survival analysis because we're mostly looking

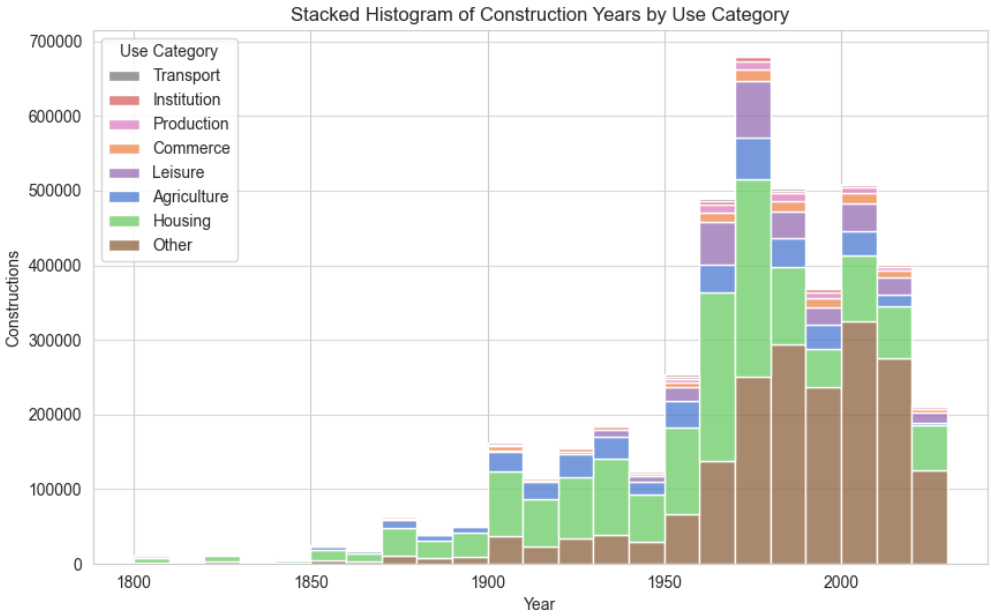

Figure 10: Building construction by usage category from 1800 to 2025. The stacked histogram shows the evolution of construction activity across eight building types. An increase in total construction volume is evident from WWII onward, with housing construction (outside others) representing the largest component of building activity in recent times.

at relatively young buildings. This clustering might bias our survival estimates, especially for older building methods and materials that don't appear much in our data.

We now turn our attention to Figure 11, which shows demolition patterns from 2000 to 2025. Demolitions increase steadily over time, probably because more buildings are reaching end-of-life and because record-keeping has improved in the BBR system. "Other", "Agriculture", and "Housing" buildings make up the most demolitions, which matches their large share of all buildings.

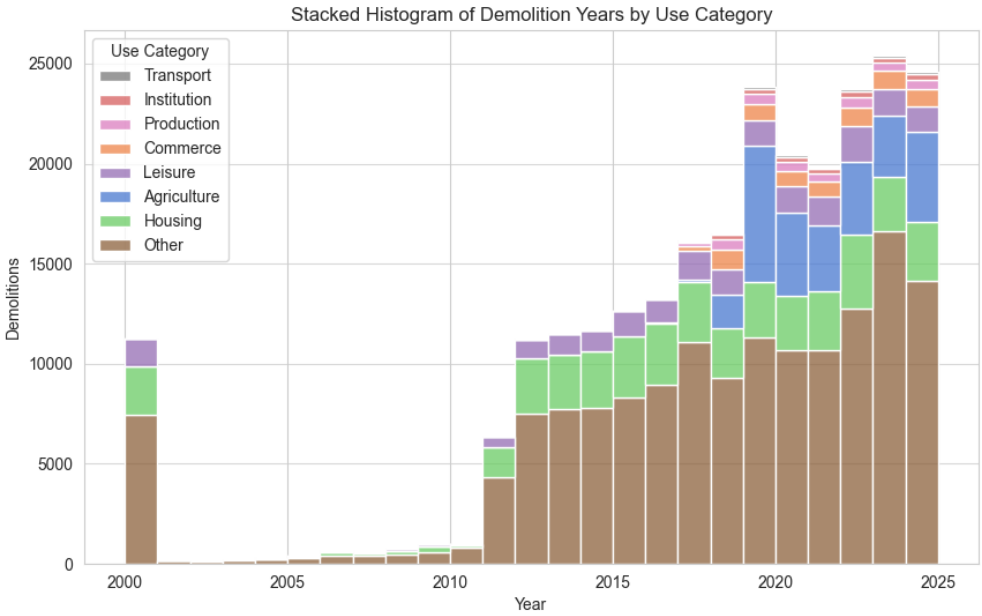

Figure 11: Annual building demolitions by category from 2000 to 2025. The sharp increase after 2017 likely reflects improved record-keeping in the BBR system rather than actual demolition growth. "Housing" and "Other" structures represent the majority of demolitions.

We believe the size of "Other" category is reasonable, since these types of buildings typically don't last as long and are easier to tear down and replace than main buildings. The category also includes run-down buildings and unknown buildings, suggesting some demolitions involve structures in poor condition or with bad records.

A *really* odd observation from the plot is the large spike in recorded demolitions around 2000, followed by an almost complete absence of demolitions from 2001-2011, before demolitions resume in 2012 and steadily increase thereafter. When we examine this closer for example by looking at the percentage of demolished buildings vs total constructed buildings in the period (see Appendix A.8), we also see this odd behaviour – a reasonable rate in 2000, then almost nothing from 2001-2011, and then again the reasonable increasing rate from 2012 onwards. However, 2012 also appears anomalous, with roughly half the demolitions recorded compared to the years 2013-2016. While we don't have clear evidence for why this occurs, Regeringen and KL (2012) notes that 2013 was when BBR data became publicly accessible, which may explain these data irregularities.

Another observation we make here is that the apparent increase in demolitions

over time, particularly the jump around 2017. Not only is it going up in number of demolitions, but new categories emerge. Prior to 2017, there are no demolitions with categories other than "Housing", "Leisure", or "Other". Rather than an actual surge in demolition activities and only beginning to tear down "Commerce", "Agriculture", etc. from this point on, it is largely due to the aforementioned system change. We believe that before 2017, demolitions were either not recorded, recorded under different classification systems, or done less systematically. Housing buildings show the most consistent demolition records throughout the period, likely because residential demolitions were prioritized in the older recording system. While we can't be completely sure, this explanation for the 2017 jump seems most likely given the visualization.

Given these inconsistencies in demolition recording before before 2013, and the administrative data collection problems of categories between 2013-2017, we decide to remove observations with demolition years prior to 2017 from our analysis, which we will expand on in Section 4.4. This ensures our models are trained on more reliable data and reduces the risk of learning from administrative errors rather than actual building patterns.

Figure 12 shows how long buildings last across all types. The distribution leans heavily toward shorter lifespans, with a long tail extending beyond approximately 150 years. A notable spike appears in the first 5 years, driven largely by the "Other" category but also with substantial contributions from "Housing" buildings.

This early peak likely reflects data quality issues rather than genuine short lifespans. Many of these very short lifetimes may represent erroneous registrations that should have received a `Status` = 11 (indicating registration errors) but were instead marked with `Status` = 10 (our chosen demolition indicator). Such errors could occur when buildings are incorrectly registered and then quickly "demolished" in the system to correct the mistake.

Excluding the problematic first 5 years, the distribution shows a more interpretable pattern. Building lifetimes increase steadily until around 65 years, where the frequency peaks before declining rapidly. The distribution then flattens between 80-125 years, maintaining relatively steady frequencies, before dropping off again and gradually tailing out. This pattern better reflects what we might expect of actual building life cycles, where most demolished buildings fall within expected service life ranges, while exceptional cases survive much longer.

While it is important to consider the big picture of building lifetimes, the "Other" category skews the visualization, and thus we also show the lifetimes with this category removed in Figure 13.

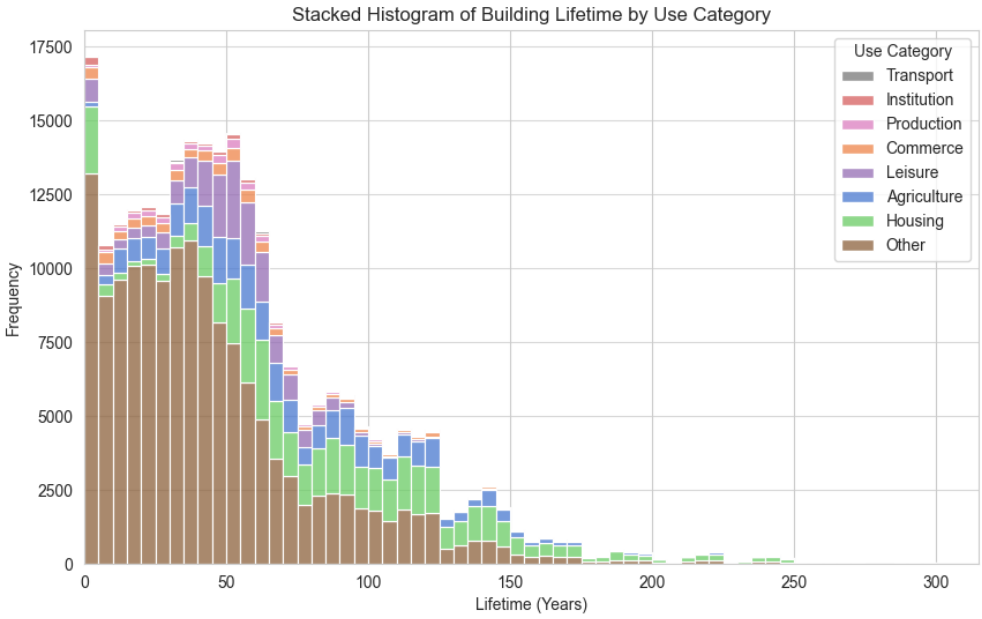

Figure 12: Distribution of building lifetimes by category. Most buildings are demolished within 80 years, with a notable spike in the first 5 years likely due to registration errors. The long tail extends beyond 150 years for some structures.

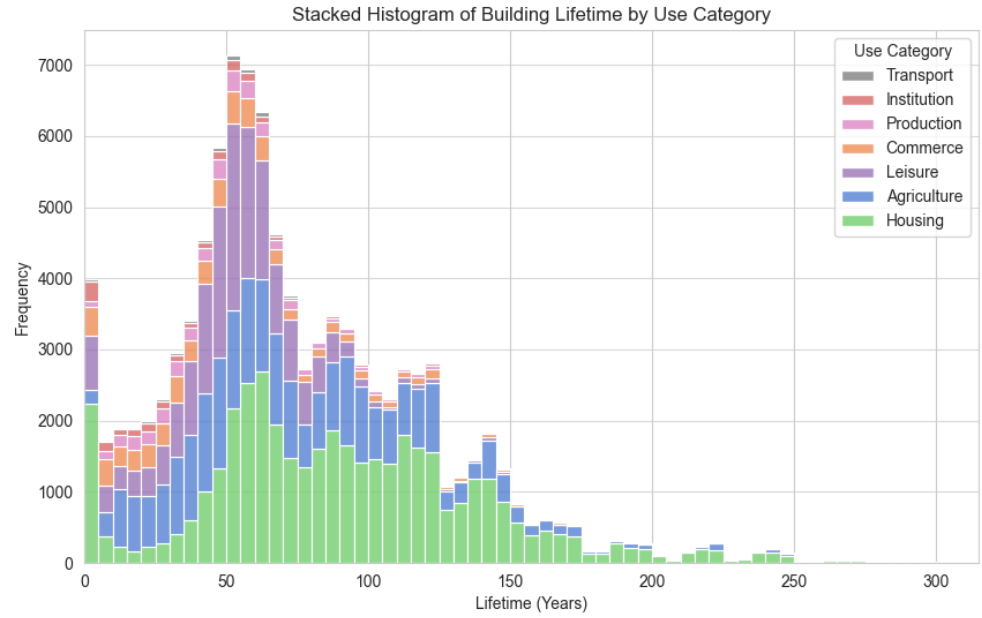

Figure 13: Building lifetime distribution excluding the "Other" category. Housing demolitions peak around 50-80 years, while agriculture and housing buildings show the longest lifespans exceeding 150 years

This distribution shows some more granular building life cycle patterns. Housing buildings dominate the middle-age demolitions, showing a broad distribution centered around 50-80 years. Agricultural buildings appear consistently throughout the lifetime range. Leisure buildings peak substantially around 45-70 years of lifetime.

We emphasize how it is mainly "Housing" and "Agriculture" that have lifetimes of more than 150 years. This might be because these building types are constructed with more durable materials and robust designs intended for long-term use, unlike buildings in the "Other" category that might be built for shorter-term purposes.

Figure 14 compares lifetime distributions across building types using box plots, which complement the histogram analysis by giving us a different view at the data structure. While the stacked histogram in Figure 12 and Figure 13 shows the overall volume and shape of lifetime distributions, the box plots provide us with statistical summaries and highlight potential outliers.

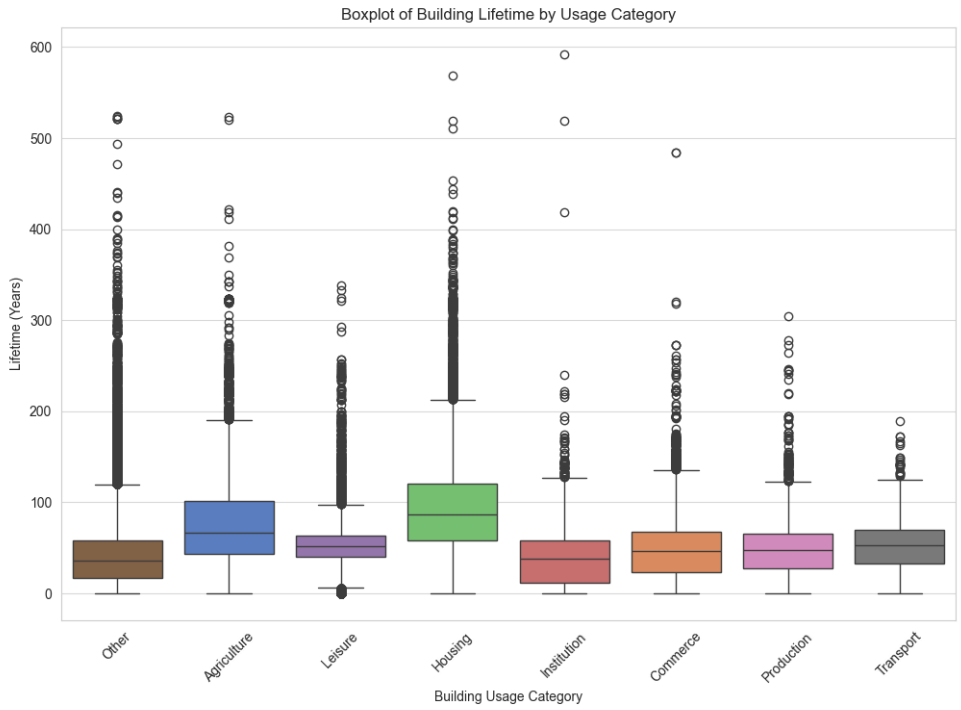

Figure 14: Box plots of building lifetimes by building category

Some key patterns become clear through this approach, and when we look at the quantified results in Table 5.

"Other", "Agriculture" and "Housing" show similar patterns – A relatively even IQR but with a wide spread. Specifically, "Other" has a median of $\tilde{x} = 36$ years with IQR = 41 years and maximum of 524 years; "Agriculture" has $\tilde{x} = 67$ years with IQR = 59 years and maximum of 523 years; while "Housing" has the highest median at $\tilde{x} = 87$ years with IQR = 62 years and maximum of 569 years. The high median of houses could be explained by the fact that people maintain homes longer because of emotional attachment and high replacement costs. The "Others" category is a catch-all for auxiliary buildings/structures. We assume most are cheap, disposable buildings (garden sheds, carports) that bring the median down. But it also includes historical unclassified buildings with uncertain purposes, which might explain the large amount of buildings outside of the whiskers.

"Leisure", "Institution", "Commerce", "Production", and "Transport" on the other hand seem to be more clustered with narrow boxes and fewer buildings with values extending beyond the whiskers – 1.5 times the inter-quartile range from the first and third quartiles. The reason for this might be because of better data quality for these categories. We say this, because if we return to Figure 11, we see how these categories generally do not have demolitions before 2017, and we have previously established that the data quality for demolished buildings is incomplete before 2017.

Use Category	Count	Mean	Std	Min	25%	50%	75%	Max
Transport	710	55.52	32.83	0	33	53	70	189
Institution	1,767	43.47	42.14	0	12	38	58	592
Production	3,446	51.48	33.90	0	28	48	66	304
Commerce	6,267	52.44	40.21	0	23	47	68	484
Leisure	18,922	52.19	26.41	0	40	52	63	338
Agriculture	27,282	75.20	44.81	0	43	67	102	523
Housing	42,762	91.97	51.57	0	58	87	120	569

Table 5: Statistical summary of building lifetimes by usage category. Here Std stands for Standard Deviation, Min and Max for the minimum and maximum values observed, and 25%, 50% and 75% for the respective quartiles.

As seen in Table 5, the standard deviations of mean lifetimes across different `Use Category` values suggest that this feature may not be a strong standalone predictor of building lifetime. The large variations observed within categories are likely driven by confounding factors. That is, the apparent differences in lifetime between `Use Category` types may largely be explained by other variables that correlate with both use type and demolition risk.

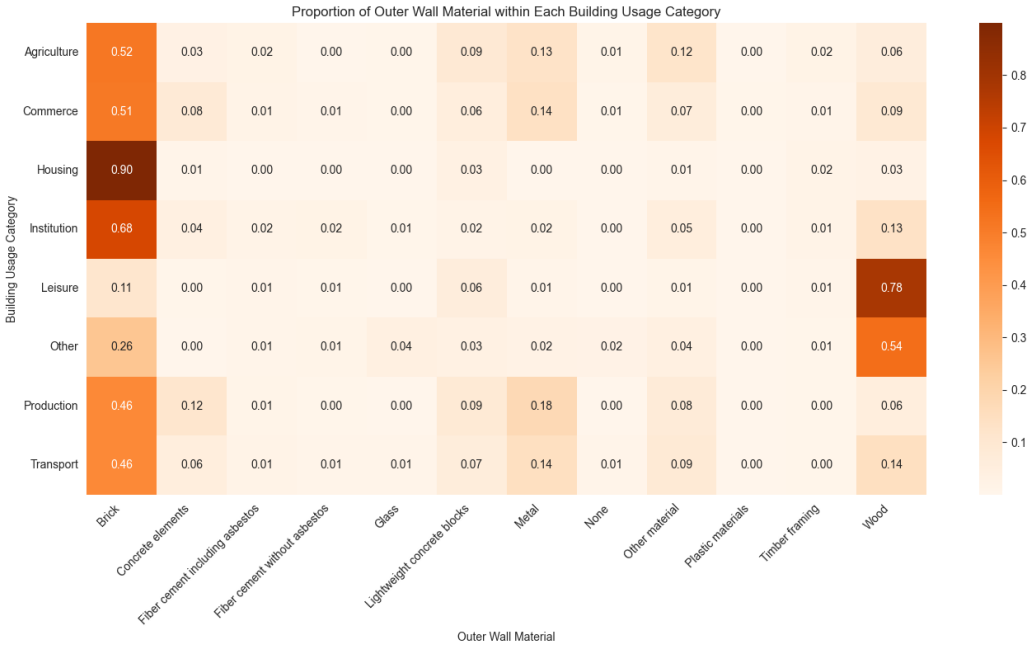

Figure 15: Heat map showing the proportion of `Outer Wall Material` within each `Use Category`. Each cell indicates the relative frequency of a given `Outer Wall Material` within a `Use Category`. For example, 90% of housing buildings have brick outer walls, while 78% of buildings in the leisure category use wood.

This is further illustrated in Figure 15, where we observe notable differences in the distribution of `outer wall material` type across `Use Categories`. For example, brick is prevalent in Housing, while wood appears frequently in Leisure buildings. These patterns suggest that some of the variation in lifetimes across `Use Category` types may be influenced by correlated features such as construction material. While we do not establish any causal relationships here, these findings indicate that `Use Category` may act as a proxy for other underlying factors that impact building lifetime.

Impact of Materials on Lifetime Trends The dataset contains the features `Outer Wall Material` and `Roof Material`, which, as the names suggest, describe the primary materials used in the outermost walls (not necessarily the load bearing walls) and in the roof. It seems highly likely that these two variables may be good predictor of building life times and we will thus explore them in more detail here.

As a first inspection, we plot a heat map of the roof and wall materials, this is shown in Figure 16 (the full translation key from number to material can be found in Appendix A.9. Looking at the heat map, some trends become apparent. Noticeably, timber framing and roof material thatch seem to correspond to the highest mean

lifetimes with their intersection having the highest value for all combinations of 179 years. There are multiple possible explanations for this, the most obvious one being that timber houses and thatch roofs simply are very durable. It is important, however, to remember the survivorship bias of our data. Since we only have demolitions from 2000-2025 and reliable demolition data from 2017-2015, it is a very short and recent time period that all recorded demolitions fall into. It is therefore feasible that many older types of buildings (such as timber houses with thatched roofs) are recorded as having lifetimes longer than what is actually true since only the long living ones survived till the present day with the opposite being true for more modern building types (see Section 4.3.2 for more information on survivorship bias).

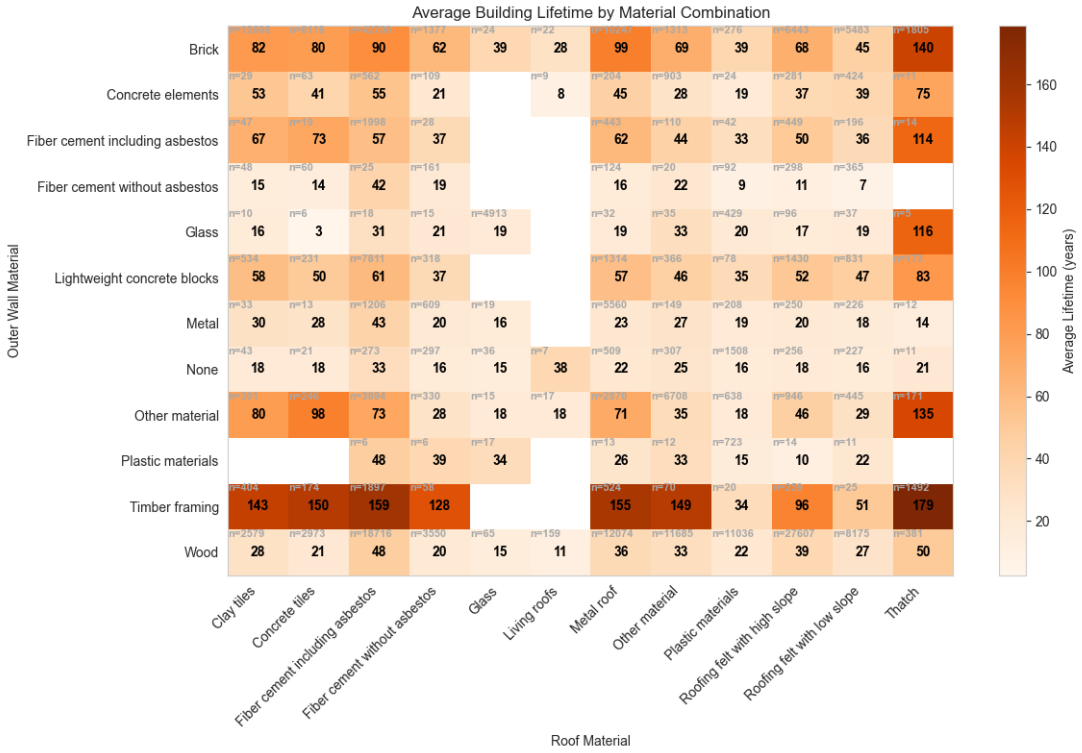

Figure 16: Heat map of the different material combinations

Some of the changing trends of which materials are being used in construction over time can be seen on Figure 17 showing the wall materials of buildings constructed from the year 1800 and later. We notice a clear trend of more and more buildings being built from wood from the 1960's and onward while brick on the other hand seems to be falling out of favor since its peak in the 1970's.

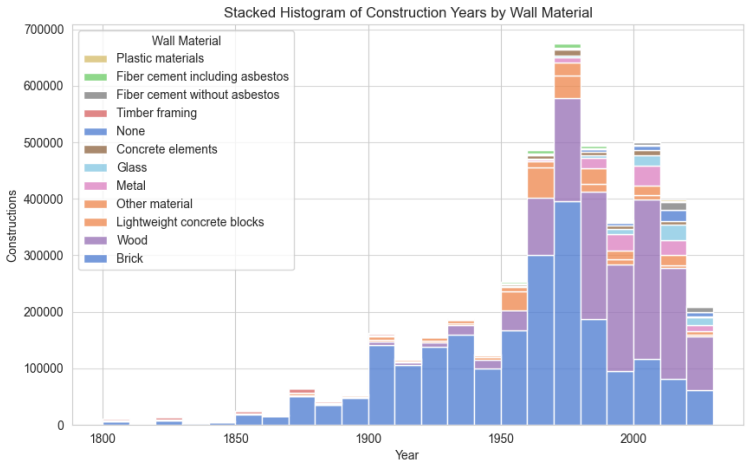

Figure 17: Histogram of constructions over time stacked by wall material

Similarly, on Figure 18 we see changing building trends when it comes to roof material. Fiber cement including asbestos was very popular until the 1980's where its prevalence started decreasing sharply. This decline was driven by the discovery of negative health impacts of asbestos and later bans of the material, the first ban in Denmark being implemented in 1972 (Kræftens Bekæmpelse, 2024). We also notice that building roofs with plastic materials and glass is a relatively new phenomenon.

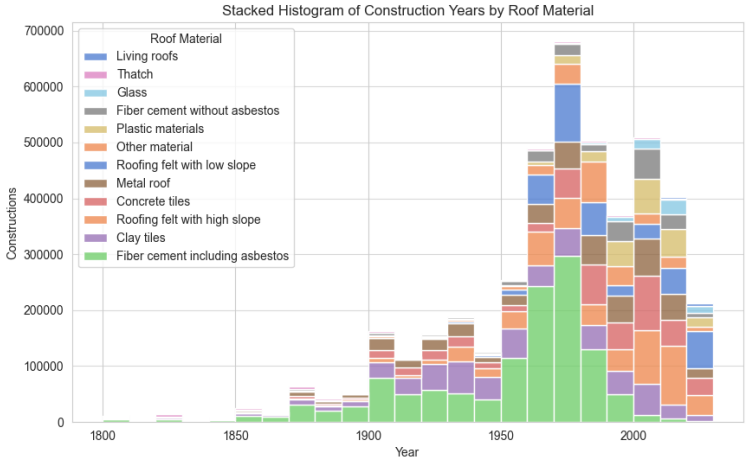

Figure 18: Histogram of constructions over time stacked by roof material

Interactive Demolition Platform To make our data analysis more accessible and give a more intuitive understanding, we created an interactive web-based map that shows all 207,958 demolitions with valid location data across Denmark. There are 45,563 with faulty coordinates. The platform provides information for each building including when it was built, when it was demolished, how long it lasted, and what materials were used.

The interactive map uses the coordinate data (feature `Coordinate`) from the dataset to show the exact location where each building was demolished. We converted these coordinates into a format that works with web maps (longtitude and latitude), allowing users to see precise demolition locations. The map automatically groups nearby demolitions together when viewing large areas, but shows individual buildings when users zoom in to street level. Specifically, we used Python's `folium` library, which gives us web-based mapping capabilities through the `leaflet.js` integration.

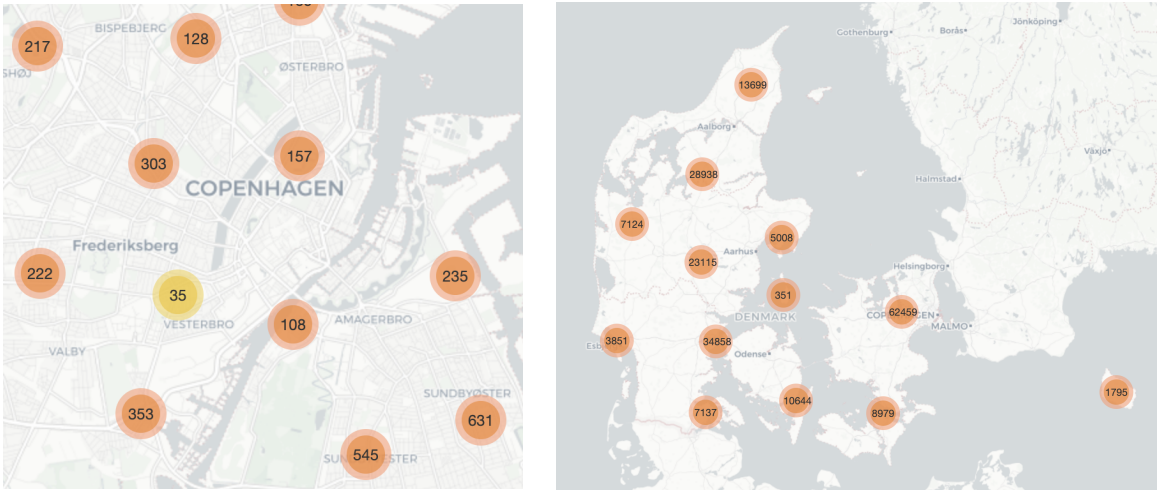

Figure 19: Maps showing the demolished buildings at different geographical scales

We hope users can engage with this website to get a more interactive and intuitive understanding of demolition patterns than traditional static plots can give. Of course, we emphasize that this interactive tool should be used in combination with traditional statistical plots and analyses to also have the quantitative understanding of the data.

The map can be **tried out online here**. Please allow 1-2 minutes of loading time, as the page isn't optimized for speed, and is hosted on a free service, Netlify.

4.3 Data Quality Assessment

Based on findings on the dataset from the exploratory data analysis section, as well as domain knowledge, we will now assess the quality of the dataset. This step is essential to avoid drawing misleading conclusions in on the findings in the later sections.

4.3.1 Censoring and Consequences

A defining feature of this dataset is the overwhelming proportion of right-censored entries, where the demolition event has not yet occurred. In our case, the majority of buildings remain in use as of the dataset's cutoff (2025), and so their true demolition times T are unobserved. We only know that $T \geq 2025$. The extent of censoring is:

- **Censored**: 4,182,757 buildings (94.29%)
- **Uncensored**: 253,521 buildings (5.71%)

This imbalance introduces some inherent challenges to the survival analysis. However, the censoring mechanism is primarily administrative (Shedden and Mesner, 2025). This comes from the fixed endpoint of our data collection rather than building-specific factors. This allows us to assume non-informative censoring, meaning that the probability of a building being censored is independent of its underlying survival time.

Tests show us that while censoring rates vary across features – ranging from 92.1% to 97.2% in building usage, and similar ranges for other characteristics – these ranges align with what we expect of building lifecycles and material durability patterns. The variation spans only 5-7% across major categories. This suggests that the censoring process does not systematically bias the sample but rather reflects natural differences in building survival. This administrative censoring is 'fine to use' with survival analysis methods, which are designed to handle such time-to-event data with incomplete end-dates.

4.3.2 Survivorship bias

A key threat to validity of our conclusions is survivorship bias, a form of sampling bias that arises when only individuals who meet certain criteria are observed. In our case, that criterion is having survived at least until the beginning of data recording. This results in left-truncation of the data, meaning that buildings demolished before the beginning of data recording are excluded from the data.

To illustrate, suppose we observe that buildings with thatched roofs have very long lifetimes. A naive interpretation might suggest that these thatched roof buildings

are exceptionally resilient. However, most thatched roof buildings were constructed long ago, and many were likely demolished before recording began. These earlier demolitions are unobserved, so our dataset only includes the subset of thatched roof buildings that happened to survive into the modern registry. This gives a misleading impression of the lifetimes of these buildings, because we are only seeing the *survivors* of a much larger original population.

This creates a biased sample. The old buildings in our dataset aren't typical for their groups; they're the ones that were most durable, well-kept, or simply "lucky". When our model learns from this data, it might wrongly infer, for example, that "buildings from the 1700's are very sturdy and less likely to be demolished" since the only buildings from the 1700's in our dataset are the ones that survived at least until we started recording.

4.3.3 System issues

In addition to quality issues we have as a results of the time the dataset spans, there are also data issues because of the collection system itself. The BBR dataset relies on building owners to register their own property information, which creates several possible sources of error. Firstly, building owners are not required to fill in all information about their building. This leads to several buildings in the registry missing values for several features. Secondly, the owner of a home might misunderstand the BBR system, and fill in information in the wrong place, or report incorrectly. Thirdly, the information that the home owner gives is not verrified. This means that we cannot trust the data in the dataset either. According to a questionnaire in an article from DR, 2023, only 6% of property inspectors believe you can trust the information in BBR. While this does not immediately quantify the error in BBR, it does highlight concerns about the overall reliability of the data.

In addition to the possible errors accuring because of self reporting of the information, the BBR system itself has some issues. One of the main issues we have found is the frequent updates to the system. For instance, the system and data structure was updated in both 2017, as well as in 2013, Regeringen and KL, 2012. In each of these updates all data should have been migrated to te new system, our experience is that this has not happened, leading to data being in different formats in the dataset. Some of these issues become apparent in Figure 11, where we can see different categorie appearing at different times.

4.4 Data Preparation for Model Training

After having cleaned and merged the BBR and supplementary datasets (see Section 4.1), additional preprocessing steps were carried out to make the data ready for

use as input to the models. We began by selecting a subset of variables for modeling building service life. These included `Region`, `Building Use`, `Outer Wall Material`, `Roof Material`, and `Footprint`. These features are the ones we have explored in Section 4.2, and therefore the one we have the best overview of. Because of the data quality issues discussed in Section 4.3, we decided not to train models on features we know very little about. All rows with missing values in any of these columns were removed to ensure clean input to the models.

Next, we added a binary indicator `Renovated` indicating if a building has been renovated (1) or not (0). This indicator is based on the feature `Year of Renovation or Extension`, which keeps the year of the latest renovation. If there is a year here the indicator is set to 1, and if there is no year, (i.e. null), the indicator is set to 0.

Then categorical variables – namely `Region`, `Use Category`, `Outer Wall Material`, and `Roof Material` – were transformed using one-hot encoding. This converts each category into a separate binary indicator, enabling compatibility with modeling methods such as the Cox model and Random Survival Forest, which require numerical input. When doing this one column was dropped from each feature to avoid perfect colinearity while preserving all information.

The feature `Footprint` is building's ground-level area or floor area – essentially the area that the building covers at ground level, measured in square meters. This feature was normalized by dividing all values with the largest recorded value for that feature. This brings the scale of the feature down to the interval $(0, 1]$, making it easier to compare with the other features, which are on the same scale.

We then computed two core variables required for survival analysis: `Lifetime` and `Event` indicator. `Lifetime` indicates the time from construction year to demolition year for demolished buildings ($\delta_i = 1$), and construction year to current year (2024) for censored buildings ($\delta_i = 0$) – essentially the age of the building. 702 buildings with negative values in `Lifetime` were removed from the dataset. `Event` indicator is defined as 1 if the building was demolished, and 0 if it was still standing at the end of 2024 (i.e., right-censored).

After having explored and assesd the data in Section 4.2 and Section 4.3 we also decide to remove demolitions from before 2017. This is largly due to the findings in Figure 11, where we can see that a different system for building categories were used before 2017, and we have – what appears to be – administrative issues before 2013. We also remove demolitions from 2025, as this year is not over yet.

Applying these steps finally leaves us with a dataset $\mathcal{D}$ with $n = 4,307,475$ observations and 4 one-hot encoded features. This dataset is presented in Table 6.

Feature	Level of Measurement	Type
Outer Wall Material	Nominal	One-hot-encoded Binary
Roof Material	Nominal	One-hot-encoded Binary
Region	Nominal	One-hot-encoded Binary
Use Category	Nominal	One-hot-encoded Binary
Footprint	Ratio	Integer
Lifetime	Ratio	Integer
Event Indicator	Nominal	Binary

Table 6: Overview of features in the dataset, including their level of measurement and data type.

5 Traditional Survival Analysis Methods

Traditional survival analysis methods form the bedrock for studying time-to-event data. These classic statistical approaches, including the Kaplan-Meier estimator, Nelson-Aalen estimator, and the Cox Proportional Hazards model, are absolutely necessary for estimating survival probabilities, cumulative hazards, and assessing covariate effects. Understanding them is fundamental before we move on to more advanced machine learning methods in survival analysis.

5.1 Survival Models Without Covariates

Survival models without covariates attempt to understand survival times when not looking at how other factors might influence them. They help describe the basic patterns of survival and the risk of an event occurring over time, even when censoring is present.

5.1.1 Kaplan-Meier Estimator

The Kaplan-Meier estimator, as introduced by Kaplan and Meier (1958), is a non-parametric approach to estimating the survival function of lifetime data and accommodates right-censored data. This makes it especially useful in cases where the event of interest (e.g., demolition) has not yet occurred for all subjects (e.g., buildings) in the sample. The Kaplan-Meier estimator is defined as:

$$\hat{S}(t) = \prod_{i=0}^{t} \left(1 - \frac{d_i}{n_i}\right) \tag{19}$$

where $t \in \{0, 1, 2, \ldots\}$ denotes the age of a building in full years , d_i is the number of buildings demolished at age i, and n_i is the number of buildings still standing (i.e., at risk) just before reaching age i.

As can be seen from Equation (19), the Kaplan-Meier estimator constructs the estimated survival function $\hat{S}(t)$ by multiplying the conditional survival probabilities at each age up to and including t. The estimate is stepwise constant, non-increasing, and defined only at integer values of t.

Derivation of the estimator By definition of the survival function, see Section 2.1.1,

$$S(t) = P(T > t)$$

where T, as before, is the lifetime of a subject, and t, with slight abuse of notation, is a observation time. For any $j = 1, 2, ..., m$, where m is the last observed event time,

$$S(t_j) = P(T > t_j).$$

We know if an event had not occurred for a subject by time t_j, it had not occurred by time t_{j-1} either. Thus, we know that the set of subjects with $T > t_j$ is included in the set of subjects with $T > t_{j-1}$, mathematically, $\{T > t_j\} \subseteq \{T > t_{j-1}\}$. Thus, since it follows from simple set theory that $A \subseteq B$ leads to $A \cap B = A$,

$$S(t_j) = P(T > t_j) = P(T > t_j \cap T > t_{j-1}).$$

Using the product rule of probability theory (Zellner, 2007) we then get,

$$S(t_j) = P(T > t_j \cap T > t_{j-1}) = P(T > t_j \mid T > t_{j-1})P(T > t_{j-1}), \tag{20}$$

where we define $t_0 = 0$ and $P(T > 0) = 1 - P(T = 0) = 1 - \frac{d_0}{n_0}$, where d_0 is the number of events at $t = 0$ and n_0 is the total number of subjects. Iterating over Equation 20, we get that,

$$S(t_j) = P\left(T > 0\right) \times \prod_{i=1}^{j} P\left(T > t_i \mid T > t_{i-1}\right). \tag{21}$$

Equivalently, we can write,

$$P\left(T > t_i \mid T > t_{i-1}\right) = 1 - P\left(T = t_i \mid T \geq t_i\right),$$

where the index i has been changed for simplicity. Thus,

$$S(t_j) = P\left(T > 0\right) \times \prod_{i=1}^{j} \left(1 - P\left(T = t_i \mid T \geq t_i\right)\right). \tag{22}$$

Our goal is now to estimate the terms $P\left(T = t_i \mid T \geq t_i\right)$. We define

- n_j as the number of subjects s at risk just before t_j. In other words,

  $$n_j = \#\{s : T_s \geq t_j\}.$$

  If a subject is censored at t_{j-1}, it is not included in n_j, as we do not know if the subject is at risk longer.

- d_j as the number of observed failures exactly at t_j. In other words,

  $$d_j = \#\{s : T_s = t_j\}.$$

  If a subject is censored at time t_j it is not included in d_j.

Using n_j and d_j, we can see that out of the n_j subjects that survived up until time t_j, d_j experience the event at time t_j, while the remaining $n_j - d_j$ survive past t_j. Thus the probability of being demolished at t_j can be empirically estimated as the number of buildings demolished at t_j over the number of buildings standing at the beginning of t_j,

$$P\left(T = t_i \mid T \geq t_i\right) \approx \frac{d_i}{n_i},$$

and hence,

$$1 - P\left(T = t_i \mid T \geq t_i\right) \approx 1 - \frac{d_i}{n_i}. \tag{23}$$

We can now plug this empirical estimate into Equation 22, and get that,

$$\hat{S}(t_j) = P\left(T > 0\right) \times \prod_{i=1}^{j} \left(1 - \frac{d_i}{n_i}\right).$$

As we saw earlier we also know that $P\left(T > 0\right) = 1 - \frac{d_0}{n_0}$, hence we can substitute this in, and pull the term into the product, we get,

$$\hat{S}(t_j) = \left(1 - \frac{d_0}{n_0}\right) \times \prod_{i=1}^{j} \left(1 - \frac{d_i}{n_i}\right) = \prod_{i=0}^{j} \left(1 - \frac{d_i}{n_i}\right),$$

which is equivalent to the Kaplan-Meier estimator from Equation 19.

Applying the Kaplan-Meier Estimator to the Building Dataset The Kaplan Meier estimator builds a stepwise estimate of the survival function $S(t)$ by multiplying the conditional survival probabilities at each observed time. In our case each unit of time is one year since construction of the building, so $\hat{S}(t)$ is piecewise constant on the intervals $[0, 1), [1, 2), [2, 3), \ldots$

At t, $\hat{S}(t)$ jumps down by the fraction $\frac{d_t}{n_t}$, where d_t is the number of buildings that were demolished at age t, and n_t is the number of buildings that were still at risk at age $t - 1$.

Figure 20 shows the Kaplan-Meier estimate for the cleaned dataset $\mathcal{D}$ defined in Section 4.4. The 95% confidence intervals are computed using Greenwood's variance and the complementary log-log transform (Williams, 1995), as implemented in the `lifelines` package. Looking at the survival function in Figure 20, we can see that it, as expected, is stepwise continuous and non-increasing. There are however other parts of the graph that seem quite odd. Taking the function at face value, one might conclude that Danish buildings have an expected lifetime of around 600 years. We find this very unlikely, given the review of previous research from Section 2.2. As

discussed in Section 4.3.2 our dataset suffers from survivorship bias. For old buildings, our dataset only includeds those that have survived for a long time. This results in undercounting both in the set d_j and n_j, but as n_j is a larger set, the fraction would become larger if we had true data. We therefore expect the true survival function to decrease more rapidly initially than what our current estiamte does, leading to the expected survival time to be shorter. In addition, we can see that the confidence intervals become quite large after around 500 years. This is because there are very few buildings left in the risk set, so the estimates become very uncertain.

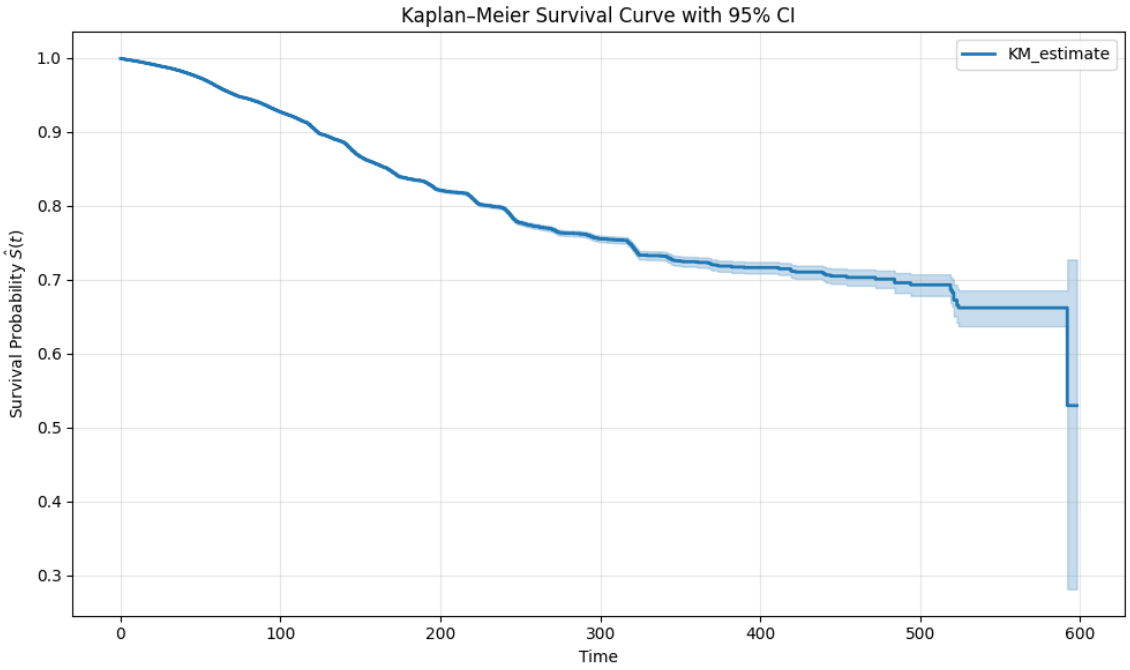

Figure 20: Kaplan-Meier estimate for the cleaned dataset with 95% confidence intervals

While the Kaplan-Meier estimator is itself a strong tool in survival analysis, it can also be used as part of other more advanced methods. The estimator does not use covariates, but it can be used in models that do, it can for example be implemented in the leaf notes of a Random Survival Forest (RSF), as will be demonstrated later in this thesis.

5.1.2 Nelson-Aalen Estimator

Complementary to the Kaplan-Meier estimator, the Nelson-Aalen estimator provides a non-parametric method for estimating the cumulative hazard function $H(t)$ from

observed time-to-event data. The estimator was introduced by Nelson (1972) and later extended by Aalen (1978). While the Kaplan-Meier estimator focuses on the probability of survival, the Nelson-Aalen estimator quantifies the accumulated risk of an event occurring over time.

The Cumulative Hazard Function (CHF), $H(t)$, represents the total yearly risk accumulated up to age t. Unlike the survival function, which is a probability bounded between 0 and 1, the cumulative hazard function is a non-decreasing function that can take any non-negative value. The Nelson-Aalen estimator estimates this function as a sum of discrete hazard contributions at each observed age. It is defined as:

$$\hat{H}(t) = \sum_{j=0}^{t} \frac{d_j}{n_j} \tag{24}$$

where $t \in \{0, 1, 2, \ldots\}$ denotes the building age in full years since construction, d_j is the number of buildings demolished at age j, and n_j is the number of buildings still at risk just before reaching age j.

Derivation of the estimator The derivation of the Nelson-Aalen estimator is similar to the derivation of the Kaplan-Meier estimator in Section 5.1.1. As seen in Section 2.1.1 the CHF, $H(t)$, is the sum of the hazards $h(t)$ up until t. The hazard/risk of an event happening at time t is conditional on the subject having survived up until just before time t. For time t_j, $h(t)$ can thus be written as,

$$h(t_j) = P(T = t_j \mid T \geq t_j),$$

and $H(t)$ as,

$$H(t) = \sum_{j=0}^{t} P(T = t_j \mid T \geq t_j). \tag{25}$$

We define n_j and d_j as in the derivation of the Kaplan-Meier estimator. We there also found that

$$P(T = t_j \mid T \geq t_j) \approx \frac{d_j}{n_j}.$$

We can plug this in to Equation 25, and get,

$$\hat{H}(t) = \sum_{j=0}^{t} \frac{d_j}{n_j}, \tag{26}$$

which is equivalent to the Nelson-Aalen estimator defiend in Equation 24.

Applying the Nelson-Aalen estimator to the Building Dataset As for the Kaplan-Meier estimator, each unit of time t is one year since construction. $\hat{H}(t)$ therefore becomes a stepwise constant function, which jumps by $\frac{d_j}{n_j}$ at age j. In other words, at age j the estimated yearly hazard is $\frac{d_j}{n_j}$, and we sum over these hazards. This results in $\hat{H(t)}$ being non-decreasing.

Figure 21 shows the Nelson-Aalen estimate for the cleaned dataset $\mathcal{D}$. As expected, we can see that the estimated CHF is stepwise constant and non-decreasing. The 95% confidence intervals are now computed using Aalen's variance estimator as described in Aalen (1978), and implemented in the package `lifelines`. The graph suffers from the same issues at the Kaplan-Meier estimate in Figure 20. Our dataset is heaviliy influenced by survivorship bias, so we expect the true hazard to be substantially higher than what is shown in the figure. Again, we can see that the confidence interval becomes very large after year 500, because of the small amount of constructions with this lifetime.

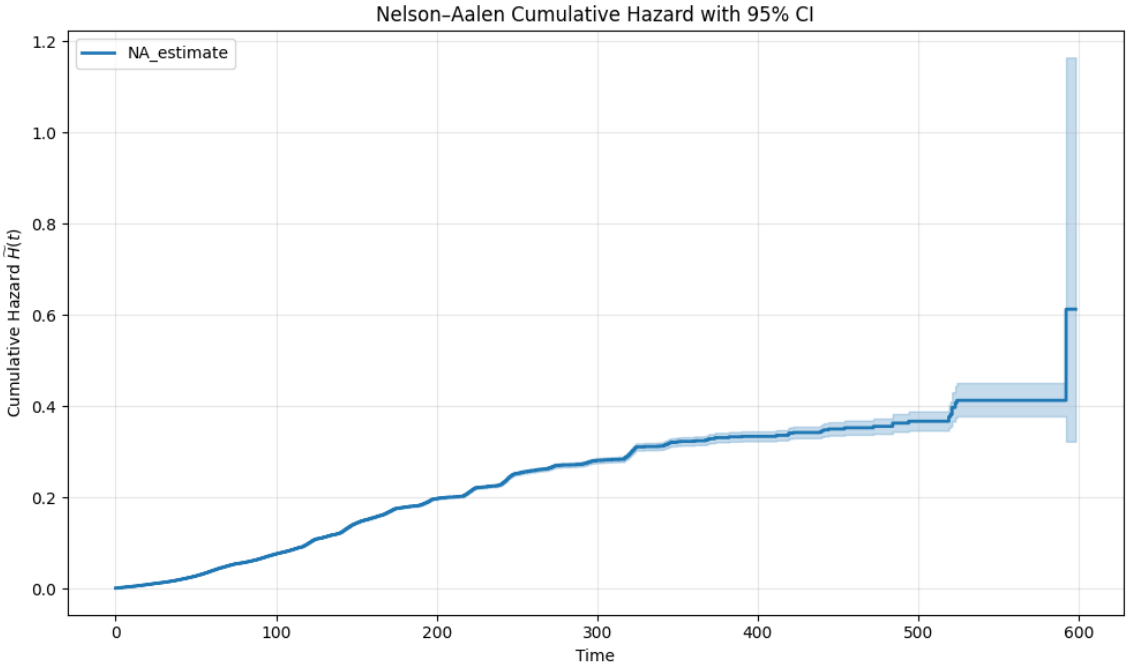

Figure 21: Nelson-Aalen estimate of the cleaned dataset with 95% confidence intervals.

As with the Kaplan-Meier estimate, the Nelson-Aalen estimator is also used as part of other more advanced methods, such as the Random Survival Forest.

5.1.3 Connecting Kaplan-Meier and Nelson-Aalen

As shown in Section 2.1.2, the relationship $S(t) = e^{-H(t)}$ follows directly from a differential equation in the continuous-time setting. However, in discrete time, this differential equation cannot be formulated: the survival function is not differentiable, and the hazard $h(k)$ is a per-period conditional probability rather than a instantaneous rate. As a result, the survival function is no longer governed by exponential decay but instead by a multiplicative update:

$$S(t) = \prod_{k=0}^{t}(1 - h(k)), \qquad H(t) = \sum_{k=0}^{t} h(k)$$

This structural difference means that the identity $S(t) = e^{-H(t)}$ no longer holds exactly in discrete time, but is instead an approximation. The Kaplan-Meier and Nelson-Aalen estimators are thus not interchangeable, as each targets a different quantity. The Kaplan-Meier estimator directly estimates the survival function $S(t)$, and we use it in the Benchmark model (introduced in Section 5.2.1), where we aim to compare survival probabilities across building categories. In contrast, the Nelson-Aalen estimator estimates the cumulative hazard $H(t)$, and is used in the Random Survival Forest model (presented in Section 6.1.1), where cumulative hazard functions are averaged across trees to form predictions.

5.2 Regression Based Survival Models Using Covariates

While univariable survival models offer a basic understanding of survival patterns, regression based survival models using covariates expand on this by incorporating external factors. These models analyze survival times by integrating other influencing variables. Specifically, we will discuss a self made benchmark model and the Cox Proportional Hazards Model, which uses a regression framework to quantify how these specific variables impact the likelihood of an event occurring or the duration of survival.

5.2.1 Benchmark Model

Before diving into more complex survival analysis methods we will introduce a simple benchmark model. This model combines Kaplan-Meier estimators for each covariate to predict the survival time of a subject.

For each one-hot-encoded feature in the dataset, (e.g. all specific materials such as brick in `Outer Wall Material` or regions such as Hovedstaden in `Region`), we identify all subjects that share it, this could for example be all subjects with brick as their `Outer Wall Material`. We fit a Kaplan-Meier estimator to each of these groups

of subjects. We then calculate the mean survival time, found as the area under the curve, for each Kaplan-Meier curve, and assign this as the expected survival time for this specific one-hot-encoded feature. In this way we give all one-hot-encoded features an expected life-time. For instance, the one-hot-encoded feature brick may be found to have an average life-time of 110 years while the one-hot-encoded feature Hovedstaden may corresponds to 45 years. For continuous features (only `Footprint` in our case) we first bin them into 4 categories using quantile-based binning, one-hot-encode them and then apply the same procedure. This approach divides the feature values at the 25th, 50th, and 75th percentiles.

Mathematically, we can create a vector $\mathbf{w}$ containing the expected lifetimes of each one-hot-encoded feature. Each element of the vector is then the mean survival time for the group of subjects belonging to that one-hot-encoded feature, calculated by taking the area under the Kaplan-Meier curve. We get that for one-hot-encoded feature k,

$$w_k = \sum_{i=1}^{m} \hat{S}_k(t_i)(t_i - t_{i-1}), \tag{27}$$

where m is the index of the final event time, and $\hat{S}_k(t)$ is the Kaplan-Meier estimator associated with feature k. The term $\hat{S}_k(t_i)(t_i - t_{i-1})$ represents the area of a rectangle, where $\hat{S}_k(t_i)$ is the height (survival probability at time t_i for k) and $(t_i - t_{i-1})$ is the width.

When predicting the lifetime of a subject we take the average of the lifetimes of the covariates present in this subject. If a subject i has a vector of covariates $\mathbf{x_i}$, we can estimate the lifetime as,

$$\hat{T}_i = \frac{\mathbf{x_i}^T \mathbf{w}}{\|\mathbf{x_i}\|}, \tag{28}$$

C-index can then be calculated based on the lifetimes of the subjects, as described in Section 3.1.1.

This model becomes a simple benchmark to compare other more complex models to. It is computationally cheap, and fully non-parametric. It also takes censoring into account through the Kaplan-Meier estimators. The model does however not look into interactions between covariates at all, and only looks at them independently. Becuase of this, the "risk" from highly correalted features can be double counted. Another downside is that we have to categorize continuous variables.

While not in this thesis's scope, the benchmark could be made even better with better parameter selection. Instead of using a fixed number of bins, we could automatically find the optimal number of bins that gives the best predictions for each

feature. Also, the area under the curve cutoff could be optimized by focusing on time periods with reliable data and reducing the influence of long time periods where the data becomes unreliable due to survivorship bias. One could also try to give the different one-hot-encoded features weights according to how much preditive power they have, or group features differently to account for correlation between them. We have however decided not to explore this further in this thesis, as we want to keep the Benchmark as a simple model.

5.2.2 Theoretical Background on the Cox Proportional Hazards Model

The Cox Proportional Hazards model (CoxPH) is a semi-parametric method for modeling time-to-event data, which estimates the *relative risk* of an observation compared to an unspecified baseline hazard (Cox, 1972). In contrast to univariable survival models, CoxPH incorporates covariates directly, allowing one to quantify how specific features (e.g., building characteristics) influence the risk of demolition. Formally, the model is defined as:

$$h(t \mid \mathbf{x}) = h_0(t) \cdot \exp(\mathbf{x}^\top \boldsymbol{\beta}), \tag{29}$$

where $h(t \mid \mathbf{x})$ denotes the hazard function at time t for an observation with covariates $\mathbf{x} \in \mathbb{R}^p$, $h_0(t)$ is the unspecified baseline hazard function, and $\boldsymbol{\beta} \in \mathbb{R}^p$ is the corresponding vector of regression coefficients. Here, p denotes the number of covariates included in the model.

In CoxPH, the objective is to estimate the regression coefficients $\boldsymbol{\beta}$ that capture the influence of each covariate on the hazard. The key insight of the model is that this estimation can be achieved without specifying or estimating the baseline hazard function $h_0(t)$, due to the assumption of *proportional hazards*. This assumption states that the ratio of hazard functions between any two individuals is constant over time. Mathematically:

$$\frac{h(t \mid \mathbf{x}_i)}{h(t \mid \mathbf{x}_j)} = \frac{h_0(t) \cdot \exp(\mathbf{x_i}^\top \boldsymbol{\beta})}{h_0(t) \cdot \exp(\mathbf{x_j}^\top \boldsymbol{\beta})} = \frac{\exp(\mathbf{x_i}^\top \boldsymbol{\beta})}{\exp(\mathbf{x_j}^\top \boldsymbol{\beta})} = \exp\left((\mathbf{x}_i - \mathbf{x}_j)^\top \boldsymbol{\beta}\right).$$

The proportional hazards assumption allows the baseline hazard function to cancel when forming relative hazard ratios, thereby enabling estimation of $\boldsymbol{\beta}$ through a partial likelihood that depends only on (time-independant) covariates.

The goal is to find the regression coefficients $\boldsymbol{\beta}$ that make the observed *order* of demolitions in the dataset most probable under the model. Let

$$t_1 < t_2 < \cdots < t_k$$

be the ordered event times. The conditional probability of observing this exact ordering of failures is

$P(i = 1 \text{ fails at } t_1 \cap \cdots \cap i = k \text{ fails at } t_k \,|\, \text{exactly one demolition occurs at each } t_1 \ldots t_k)$.

From here on the condition that exactly one demolition occurs at each t_k is assumed as a condition in each of the following probability-equations. In the discrete-time event, this naturally raises the question of how to handle tied events (when multiple events occur at some t_k) – we will touch upon this later in the section.
Using the chain rule of probability theory, (5.2.2) can be written as,

$$\begin{aligned} P(i = 1 \text{ fails at } t_1) \cdot P(i = 2 \text{ fails at } t_2 \mid i = 1 \text{ failed at } t_1) \cdot \\ P(i = 3 \text{ fails at } t_3 \mid i = 1 \text{ failed at } t_1 \text{ and } i = 2 \text{ failed at } t_2) \cdot \ldots \cdot \\ P(i = k \text{ fails at } t_k \mid i = 1 \ldots i = k - 1 \text{ failed at } t_1 \ldots t_{k-1}) \end{aligned} \quad (30)$$

Each of these probabilities are estimated using equations of the form,

$$P(i = 1 \text{ fails at } t_1) = \frac{\exp(\boldsymbol{x_1}^T \boldsymbol{\beta})}{\sum_{j \in \mathcal{R}(t_1)} \exp(\boldsymbol{x_j}^T \boldsymbol{\beta})},$$

for t_1, and more broadly for t_k,

$$P(i = k \text{ fails at } t_k \mid i = 1 \ldots i = k - 1 \text{ failed at } t_1 \ldots t_{k-1}) = \frac{\exp(\mathbf{x}_k^\top \boldsymbol{\beta})}{\sum_{j \in \mathcal{R}(t_k)} \exp(\mathbf{x}_j^\top \boldsymbol{\beta})}, \quad (31)$$

where the risk set, $\mathcal{R}(t_i)$, denotes all the subject still at risk just before t_i. Right censored data is thus handled in CoxPH by being included in the risk set in the denominator, but it will never appear in the numerator.

This formulation in Equation 31 stems from how the Cox partial likelihood is constructed. At each event time t_k, we observe that *some subject* in the risk set $\mathcal{R}(t_k)$ fails. The model then defines the probability that it was individual $i = k$ as proportional to their hazard at that time. The condition in each of the probabilities is taken account for by removing previous demolitions in the risk set. This is the core quantity used in constructing the Cox partial likelihood.

We have thus shown that at each observed event time t_i, the probability that individual i experiences the event, given that someone in the risk set $\mathcal{R}(t_i)$ does, is:

$$\mathcal{L}(\boldsymbol{\beta} \mid \mathbf{x_i}) = \frac{h(t_i \mid \mathbf{x}_i)}{\sum_{j \in \mathcal{R}(t_i)} h(t_i \mid \mathbf{x}_j)} = \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\beta})}{\sum_{j \in \mathcal{R}(t_i)} \exp(\mathbf{x}_j^\top \boldsymbol{\beta})}, \quad (32)$$

While the mathematical expression is identical to the conditional probability above, the interpretation differs: the likelihood treats the observed demolition as fixed data and evaluates how well different parameter values $\boldsymbol{\beta}$ explain this observation.

A consideration in our case is that we are working with discrete time intervals rather than continuous time, which is a complication since there is no meaningful order for buildings demolished in the same year. One common way to address this issue is to use the *Breslow approximation*, which simply assumes that all tied events at a given time share the same risk set and occur sequentially. The contribution of these tied events to the partial likelihood is then computed as the product of individual likelihood contributions, each using the full risk set at that time. This approximation is computationally efficient and widely used, and it is the default tie-handling method in `sksurv` (Breheny, 2017).

We now want to find the parameter $\boldsymbol{\beta}$ which maximizes the likelihood of the observed data. By using the chain rule from Equation 30 and taking the product over all subjects where the event was observed, i.e., those with event indicator $\delta_i = 1$, we get the partial likelihood function,

$$\mathcal{L}(\boldsymbol{\beta} \,|\, \mathbf{x}) = \prod_{i:\delta_i=1} \frac{\exp(\mathbf{x}_i^\top \boldsymbol{\beta})}{\sum_{j \in \mathcal{R}(t_i)} \exp(\mathbf{x}_j^\top \boldsymbol{\beta})}. \tag{33}$$

The coefficient vector $\boldsymbol{\beta}$ is estimated by maximizing the partial likelihood function above. However, directly maximizing a product of terms can be numerically unstable and computationally inefficient. For this reason, most software packages, including `sksurv`, instead optimize the *log partial likelihood*, which turns the product into a sum and thus eases the use of numerical methods for maximization (Piech, 2023). Taking the log of the likelikehood $\mathcal{L}(\boldsymbol{\beta})$ in Equation 33 we get,

$$\log \mathcal{L}(\boldsymbol{\beta} \,|\, \mathbf{x}) = \sum_{i:\delta_i=1} \left[\mathbf{x}_i^\top \boldsymbol{\beta} - \log \left(\sum_{j \in \mathcal{R}(t_i)} \exp(\mathbf{x}_j^\top \boldsymbol{\beta}) \right) \right].$$

Our goal is now to maximize $\log \mathcal{L}(\boldsymbol{\beta} \,|\, \mathbf{x})$, and thus find $\hat{\boldsymbol{\beta}}$, which maximizes the likelihood. Mathematically,

$$\hat{\boldsymbol{\beta}} = \arg\max_{\boldsymbol{\beta}} \sum_{i:\delta_i=1} \left[\mathbf{x}_i^\top \boldsymbol{\beta} - \log \left(\sum_{j \in \mathcal{R}(t_i)} \exp(\mathbf{x}_j^\top \boldsymbol{\beta}) \right) \right].$$

In most software implementations, including `sksurv`, this maximum is found by solving,

$$\frac{\partial}{\partial \beta} \log \mathcal{L}(\boldsymbol{\beta}) = 0,$$

and find the roots using numerical root-finding algorithms like Newton-Raphson (Therneau and Grambsch, 2000).

Each coefficient β_k represents the log hazard ratio associated with a one-unit increase in the k-th covariate, holding all other variables constant. A positive coefficient indicates that the corresponding feature increases the hazard (i.e., shortens expected survival), while a negative coefficient implies a lower hazard.

While the Cox model is widely used due to its interpretability and flexibility, it relies on the proportional hazards assumption. If this assumption is violated, for example if the effect of a covariate changes over time, then the model may perform poorly.

5.2.3 Feature Importance for Cox Proportional Hazards Model

We apply the Cox Proportional Hazards model (CoxPH) to the cleaned dataset from Section 4.4. In this section we present preliminary findings on optimal regression coefficients $\hat{\boldsymbol{\beta}}$. We go into more detail on the model's performance in Section 8.

Compared to the other models we will investigate, CoxPH is computationally cheap. Since we are now only looking into this model without comparing it to the rest, we can use the full dataset. For this analysis we perform 5-fold cross-validation, to ensure that our results are not influenced by the split we make. The results presented here are averaged across the 5 folds. For this analysis, we will use the dataset as explained in Section 4.4, and we have dropped one category from each one-hot-encoded feature. These dropped categories are what the log hazards in $\hat{\boldsymbol{\beta}}$ are compared to. The categories we have dropped are the ones that appeared first in each feature, and can be seen in the table below.

Feature	**Dropped Value**
Region	Hovedstaden
Use Category	Agriculture
Outer Wall Material	Brick
Roof Material	Roofing felt (low tilt)

Table 7: Categories removed from the dataset

The parameters presented in this section are the hazard ratios, (i.e. $\exp(\hat{\boldsymbol{\beta}})$), not the raw log coefficients. The values represent multiplicative changes in the hazard compared to the omitted categories. Here, a hazard ratio > 1 means that the category increases the risk of demolition compared to the reference, while a hazard ratio < 1 decreases the risk. For instance, if category A has a hazard ratio of 1.1, this means that the yearly risk of demolition for buildings in category A is multiplied with

1.1, compared to the reference. In Table 8 we present some of the most interesting coefficients we found. The full list of coefficients can be found in Appendix A.10

Feature	Level	Hazard Ratio
Region	Nordjylland	1.40
Region	Sjælland	0.93
Use Category	Housing	0.49
Use Category	Commerce	1.34
Outer Wall Material	Plastic cladding	10.99
Outer Wall Material	Timber framing	0.64
Outer Wall Material	Fiber cement (incl. asbestos)	3.47
Roof Material	Green roof	1.95
Roof Material	Clay tile	0.51
Footprint	Per m^2	3.93×10^{-6}
Renovated	Yes vs. No	0.49

Table 8: Selected yearly hazard ratios for key predictors (vs. their reference levels).

When interpreting the results in Table 8 it is important to remember the reference they are compared to. If we look into the `Outer Wall Material` coefficients we find that both Plastic and Fiber Cement have very high coefficients. The reference for this category is Brick, which is a material we expect to have high durability. As a result all wall materials, except for Timber Framing, have a coefficients greater than 1. We will get back to the Timber Framing coefficient shortly. This same effect can be seen for the other categorical features as well, but here the category which is dropped seems to be more standard when it comes to durability, as therefore the hazard rates are both above and below 1.

Another important consideration is that the building characteristics are correlated with each other – also known as confounding (Jager et al., 2008). For instance, the distribution of building use categories varies significantly across regions, as shown in Figure 8. Similarly, wall material choices are strongly associated with building types. Carports are commonly built with plastic materials, while apartment buildings typically use brick construction.

This correlation between features affects how we interpret the Cox model coefficients. When features are correlated, the estimated effects become mixed up, making it difficult to isolate the individual contribution of each feature. This can help explain

several patterns in our results. The very high coefficient for plastic wall material may partly reflect that plastic is mostly used for carports, sheds, etc. that inherently have shorter lifespans; the higher hazard rate in Nordjylland compared to Hovedstaden could reflect regional differences in building type composition, and the apparent importance of built area and renovation status may be confounded with building type. This could be because carports typically have small areas and are rarely renovated, while apartment buildings have larger areas and are frequently renovated.

This is however not to say that it is not interesting to look at. The correlations between material choice and building type often tell a story about rational design decisions. For instance, plastic is chosen for carports because these structures do not require the same longevity as permanent buildings.

Finally, we once again must discuss survivorship bias (see Section 4.3.2 for more). The coefficients, like all other results in our thesis, will be influenced by the survivorship bias in our data. This can be easily seen in the category Timber Framing for Wall Materials, which shows a hazard ratio of 0.64, suggesting 36% lower demolition risk. This is a type of construction that is rarely built nowadays, so the buildings that have this wall material in our dataset are typically constructed a long time ago. Since we only have the buildings in these categories that have already survived for a long time, it will seem like this wall material makes buildings have a very long lifetime. We cannot, however, say that this is the case, as a large part of these buildings have of course already been demolished a long time ago. Similarly, clay tile roofs show a hazard ratio of 0.51. This lower demolition risk could reflect multiple factors: clay tiles are typically used on higher-quality buildings built to last, though survivorship bias may also play a role since only the most durable clay-tiled buildings from earlier periods remain in our dataset.

6 Ensemble Methods for Survival Analysis

This section focuses primarily on Random Survival Forest, providing detailed theoretical background and comprehensive analysis of its application to building lifetime prediction. We also briefly explore Gradient Boosting and Component-wise Gradient Boosting methods as supplementary approaches to provide a more complete comparison of ensemble techniques for survival analysis.

6.1 Theoretical Background

Ensemble methods are a class of machine learning algorithms that combine the predictions of multiple base estimators (models) to improve the overall predictive performance, stability, and robustness compared to any single constituent model. The core idea is that by aggregating the "wisdom" of diverse models, the weaknesses of individual models can be averaged out. In this section we will introduce three ensemble methods for survival analysis.

6.1.1 Random Survival Forest

Random Survival Forest (RSF) is an ensemble machine learning method extending the random forest algorithm (Breiman, 2001) to survival analysis (Ishwaran et al., 2008). RSF builds upon the principles of bagging (Bootstrap Aggregating) and random feature selection, similar to the original Random Forest algorithm.

Decision Trees for Survival Analysis At the core of RSF are survival trees. Survival trees are modifications of standard decision trees (Quinlan, 1986) adapted for time-to-event outcomes. We remind the reader that in survival analysis, the outcome includes both an event indicator (whether the event occurred) and the age of the subject at the time of the event or censoring.

A survival tree recursively partitions the p-dimensional feature space $\mathcal{X} \subseteq \mathbb{R}^p$ into disjoint regions $R_1, R_2, \ldots, R_M$, creating nodes that group observations with similar survival characteristics. For each observation with features $\mathbf{x}_i \in \mathbb{R}^p$, the tree assigns it to a unique leaf node based on its feature values. Figure 22 shows a pseudo-visualization of how such a tree is partitioned within the RSF.

In contrast to standard decision trees that minimize impurity measures like Gini or Entropy, survival trees use statistics that quantify differences in survival distributions between nodes. Specifically for RSF the log-rank test statistic is used as the splitting criterion.

In RSF, each survival tree then estimates a Cumulative Hazard Function (CHF), $\hat{H}(t)$, or a Survival Function, $\hat{S}(t)$, for each leaf node. This is done non-parametrically

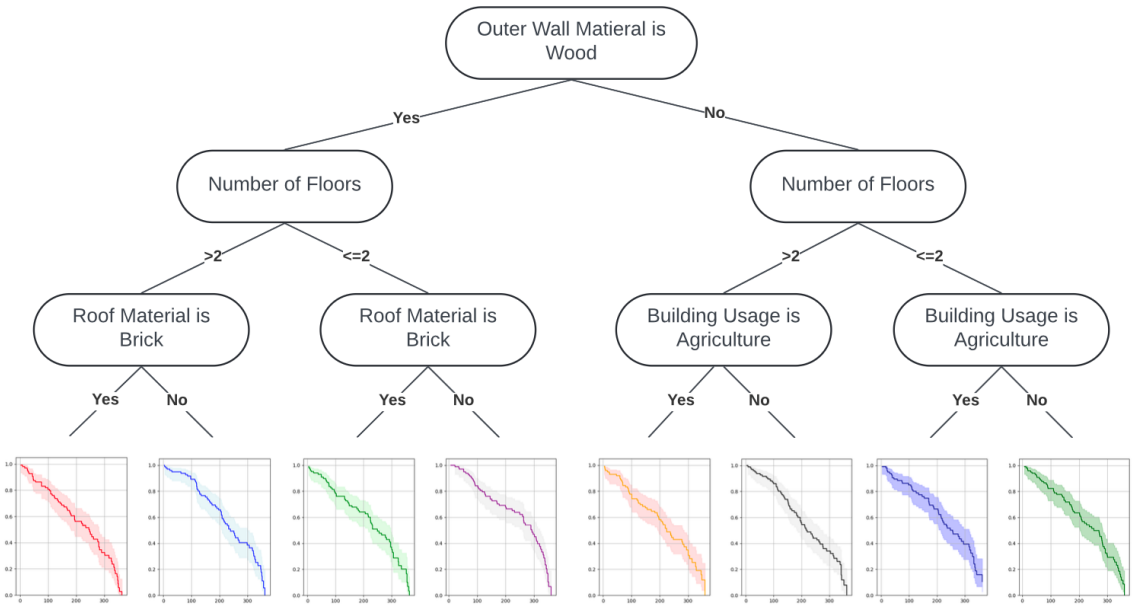

Figure 22: A pseudo-survival tree illustrating the partitioning logic within a Random Survival Forest (RSF). Features like material and usage lead to splits, with leaf nodes showing estimated survival functions.

(Section 5.1.1) from the observations that fall into that node.

The log-rank Splitting Criterion The key to building survival trees is the splitting criterion used to create branches. Ishwaran et al. (2008) use the log-rank test statistic for the splitting criterion, which measures the difference in survival distributions between two groups. This approach has been validated in the literature, as demonstrated by the survival tree methodology proposed by LeBlanc and Crowley (1993).

We consider a candidate split of node s - with n subjects - into child nodes s_L and s_R. In node s there are M ages where events - either demolition or censoring - are observed, $t_1 < ... < t_M$, the reason there are M, rather than n ages, is due to possible ties in age. For each age t_j we define:

- d_j is the number of events at t_j in the parent node
- $d_{L,j}$ is the number of events at t_j in the left child node
- Y_j is the number of subjects at risk just before t_j in the parent node
- $Y_{L,j}$ is the number of subjects at risk just before t_j in the left child node

If we assume that the two child nodes have identical hazard functions, the expected number of events at age t_j in the left child is,

$$E(d_{L,j}) = Y_{L,j}\frac{d_j}{Y_j},$$

since the hazard at age t_j is $\frac{d_j}{Y_j}$ in the parent node, and we have $Y_{L,j}$ subjects at risk in the child node. For each age we thus get a difference in observed events minus expected events of,

$$O_j - E_j = d_{L,j} - Y_{L,j}\frac{d_j}{Y_j}.$$

Summing this over all ages gives an overall difference of,

$$D = \sum_{j=0}^{M}\left(d_{L,j} - Y_{L,j}\frac{d_j}{Y_j}\right).$$

To get a fair risk score, which does not take the size of the nodes into account, we now want to standardize this difference. This is done by dividing by the variance. Each $d_{L,j}$ can be seen as a hypergeometric draw. Out of the entire population Y_j, we have $Y_{L,j}$ successes (left child members). We then draw d_j items without replacement, and want to see how many come from the success (left child) group. Using the formula for variance of a hypergeometric distribution, we therefore get the variance,

$$\mathrm{Var}(d_{L,j}) = Y_{L,j}\left(1 - \frac{Y_{L,j}}{Y_j}\right)\frac{d_j}{Y_j}\left(1 - \frac{d_j}{Y_j}\right).$$

Summing this over all ages we get,

$$\sum_{j=0}^{M} Y_{L,j}\left(1 - \frac{Y_{L,j}}{Y_j}\right)\frac{d_j}{Y_j}\left(1 - \frac{d_j}{Y_j}\right).$$

It should be noted that this summation relies on an assumption that each $d_{L,j}$ is independent, so their variance terms can be summed independently. This assumption is well established in the survival analysis literature (Fleming and Harrington, 1991).

Putting these together, and squaring the difference to avoid negatives, we get the log-rank statistic:

$$L(s, s_L, s_R) = \frac{\left(\sum_{j=0}^{M}(d_{L,j} - Y_{L,j}\frac{d_j}{Y_j})\right)^2}{\sum_{j=0}^{M} Y_{L,j}(1 - \frac{Y_{L,j}}{Y_j})\frac{d_j}{Y_j}(1 - \frac{d_j}{Y_j})} \tag{34}$$

The split that maximizes this statistic is chosen. This creates the most significant difference in survival between the resulting nodes. Intuitively, the log-rank statistic measures how much the number of events we actually see in a potential new group is different from what we would expect to see if that group's survival was exactly the same as the original, larger group. Thus, by maximizing this statistic, the survival tree ensures that each split creates two subgroups with as different hazard functions as possible.

The Random Survival Forest Algorithm The RSF algorithm combines multiple survival trees to produce a more robust and accurate prediction model. In essence, turning many survival trees into a survival forest. The algorithm can be seen in Appendix A.4.

Once the Random Survival Forest is trained, predicting the outcome for a new unseen sample means aggregating the predictions from all individual survival trees in the ensemble. The new sample is passed down each survival tree until it reaches a leaf node, which holds a CHF, estimated by the Nelson-Aalen Estimator (see Section 5.1.2). The forest-level (ensemble) CHF is obtained by averaging over all B trees:

$$\hat{H}_{\text{ensemble}}(t \mid \boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{H}_b(t \mid \boldsymbol{x})$$

This ensemble CHF reflects the estimated accumulated risk of the event having occurred by time t for a given sample $\boldsymbol{x}$.

A particularly useful scalar summary derived from the ensemble CHF is the *risk score*, which quantifies the overall level of risk associated with a sample. Although the theoretical CHF could be defined over continuous time, our analysis uses discrete time (years), so our estimated CHF is only defined and evaluated at the finite number of distinct demolition years $t_1, \ldots, t_M$ observed in the training data. The risk score is computed as the sum of the ensemble CHF values over these time points:

$$\text{Risk}(\boldsymbol{x}) = \sum_{i=1}^{M} \hat{H}_{\text{ensemble}}(t_i \mid \boldsymbol{x})$$

This scalar risk score does not correspond to a specific survival probability or time, but rather serves as a relative measure – higher values indicate a higher expected number of events and thus a greater overall risk. In practice, this is useful for ranking individuals by risk, as with Harrel's Concordance Index, which is detailed in Section 3.1.1.

6.1.2 Gradient Boosting Survival Analysis

In addition to investigating Random Survival Forests, we also want to try out some other ensemble methods, which were easily available through `sksurv`. We will not dive deep into the theoretical background of these models, but thought it would still be interesting to see how they perform for future research. We also believe this fits well into our thesis, as it is mainly a proof of concept. In this section we will briefly introduce Gradient Boosting for Survival Analysis, which is one of the two boosting methods we have tried out.

Gradient Boosting (GB) is an ensemble method initially introduced by J. H. Friedman, 2001. GB builds a strong prediction by adding many simple decision trees, where each tree now learns to "fix" the biggest errors of the previous ones (Ridgeway, 2007). When applying GB to survival analysis, we measure errors using Cox partial-likelihood loss, introduced in Section 5.2.2. This measure of loss handles both event times and censoring naturally (Hothorn et al., 2006). Very simply put, the GB algorithm goes through this process in every step, (Pölsterl, 2020b):

1. Calculates how much the current model misorders subjects by failure risk, in other words, calculates the residuals. The residuals in Gradient Boosting are found as the negative gradient of the Cox partial likelihood loss function (Ridgeway, 2007).

2. Fits a small tree to these residuals, to find where the model needs to biggest corrections.

3. Update the overall risk score for each subject by adding the tree's predictions.

After a specified number of iterations. the summed trees give each subject a log-hazard score, which we can use to calculate a C-index, as introduced in Section 3.1.1, (Pölsterl, 2020b). Findings on Gradient Boosting are presented in Section 8.

6.1.3 Componentwise Gradient Boosting Survival Analysis

Componentwise Gradient Boosting for Survival Analysis is the second boosting method we explore in this thesis. As with Gradient Boosting, we will not dive deep into the theoreticla background of this method, but provide a brief introduction here.

While traditional Gradient Boosting uses decision trees as base learners, componentwise boosting uses simple linear functions as base learners (Bühlmann and Hothorn, 2007). The goal is to build a model of the form $h(x) = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_n x_n$, where – with slight abuse of notation – $h(x)$ represents the log-hazard and $\beta = (\beta_1, \beta_2, ..., \beta_n)$ are the coefficients for the respective features. At each boosting iteration only one component of β is updated. Residuals are calculated in the same

way as for Gradient Boosting, as the negative gradient of the Cox partial likelihood loss function. Instead of fitting a tree to these residuals, componentwise boosting fits a simple linear regression, using only one feature at a time (Hothorn et al., 2006). At each iteration, the algorithm evaluates all features individually, fits a linear model between each feature and the current residuals, and selects the feature with the best fit. Only the coefficient for this selected feature is then updated, while all other coefficients remain the same.

Componentwise Gradient Boosting has the advantage of producing a linear model that is highly interpretable, while simultaneously performing automatic feature selection by choosing only the most relevant features across iterations (Pölsterl, 2020b). The resulting model typically selects only a small subset of relevant features, (Bühlmann and Hothorn, 2007). Findings on Componentwise Gradient Boosting are presented in Section 8.

6.2 Implementation and Computational Considerations

Now that we've covered the theory behind these ensemble methods, we will go over how they are actually implemented and what challenges we face when applying them to real data. This section explains the practical aspects of building and running these models, how we tune the models, and ensure our results are reliable.

As discussed in Section 3.2, each model is wrapped in a consistent class structure that standardizes the interface for training, prediction, and evaluation. All the ensemble methods in this study use the `scikit-survival` library, which is a well-tested implementations of survival analysis methods for Python.

Hyperparameter Optimization For the ensemble methods, we define hyperparameter ranges that balance model complexity with our computational constraints. The actual hyperparameter grid searches used in model training are shown explicitly in Section 8. While some hyperparameters apply to all ensemble methods, others are specific to either Random Survival Forest or the boosting approaches.

The most critical parameter for the ensemble methods is the number of estimators, which determines how many individual trees are combined together. While more estimators generally improve performance, they also increase model complexity and can lead to overfitting. We test from single trees to large ensembles to find the optimal balance between underfitting and overfitting for our building data.

Learning rate controls how fast the model learns from mistakes in boosting methods. When using many estimators, each tree attempts to correct the mistakes of previous ones. With a high learning rate, these corrections can be large and aggressive, causing the model to quickly overfit to training data noise. However, values

less than one force each tree to make only small corrections, requiring more trees to achieve the same performance but resulting in a more gradual and stable learning process. This allows us to benefit from using many estimators (which generally improves performance) while reducing the overfitting risk that typically comes with model complexity.

Tree depth constrains the maximum number of sequential splits each tree can make, which controls of course model complexity. We want to train shallow trees because deeper structures tend to overfit the training data (Maniwa et al., 2024).

Minimum sample split requires each tree node to have enough buildings before splitting. This tells each tree not to make splits that would result in predictions based on too little data. We experiment with a wide range of values here, in order to find the optimal balance between preventing overfitting and maintaining the tree's ability to discover meaningful patterns.

Dropout rate serves as a way of regularization by randomly excluding some previously fitted base learners during training. We test a range from no regularization to moderate levels, aiming to prevent overfitting. This forces each new tree to account for the possibility that some earlier predictions might be missing

Computational Considerations The ensemble methods give us significant computational challenges. Training time increases substantially with the number of estimators, where models with 1000 estimators requires a lot more training time than any other hyperparameter does. This computational burden is especially a problem for the boosting methods, which must build sequentially rather than in parallel, as we can for the Random Survival Forest.

The biggest slowdown happens when making predictions with Random Survival Forest. We experiment with training an RSF model on a basic holdout split of 20% test and 80% training, on a simple hyperparameter set (most importantly with only 10 estimators). Here, usually it takes just a few minutes to train the model, but predicting is the limiting factor. In our tests, running predictions on this split of our data took over 10 hours without finishing, even when using all 8 available cores of our laptops (`n_jobs=-1`). We're not entirely sure why this happens, since predicting should be straightforward – each prediction just traverses all trees and computes survival functions at each leaf – but this process seems to scale poorly with both forest size and dataset size. We hypothesize there is a problem with the implementation of the prediction method from `scikit-survival`, however, we haven't investigated this further.

In our full nested cross-validation training framework, most of the total 16 hours of runtime is spent on the boosting methods. Specifically, running a grid search over 5 outer folds and 5 inner folds takes roughly 16 hours end to end, and the

boosting models dominate this training time. We suspect the main reason is that the particular boosting implementation in `scikit-survival` does not expose any built-in way to split processing work across cores. In contrast, RSF trains much faster, perhaps because it can distribute work across all 8 cores. So the lack of parallelism in boosting appears to be the critical bottleneck.

The computational limitations also influenced our hyperparameter grid design, leading to relatively small parameter spaces with only 24-32 combinations per method. While more extensive grids would be nice, the practical constraints of training and prediction time make such this computationally "impossible" given our available resources.

6.3 Feature Importance for Ensemble Methods

Just as we saw in Section 5.2.3 which features matter most for predicting demolitions is interesting to analyze. Random Survival Forest makes predictions by combining many trees, but this makes it hard to see which features actually drive the decisions. To solve this, we use permutation importance – a method that tests how much performance drops when we break each feature's connection to the outcome. This is the recommended approach as by the documentation of the Random Survival Forest library we are implementing.

Permutation Importance Permutation importance measures how important a feature is by seeing how much the worse the model gets when that feature becomes random (Altmann et al., 2010). The idea is simple: if a feature really matters for predictions, then shuffling its values randomly should hurt the model's performance a lot. It works this way: for each feature, it randomly shuffles the feature's values while keeping everything else unchanged, then measures how much the model's performance drops. For a trained Random Survival Forest, we calculate importance like:

$$\text{Importance(feature)} = \text{Performance}_{\text{original}} - \text{Performance}_{\text{shuffled}}$$

Again, we measure performance using the concordance index on test data. If shuffling feature A drops the C-index from 0.80 to 0.65, then A has high importance (0.15). If shuffling feature B only drops it from 0.80 to 0.78, then B has low importance (0.02). If feature C increases the C-index, it can be interpreted as C actually hurting the model's predictive performance, which could point to confounding or simply noise.

However, most features are one-hot encoded in our dataset. For example, wall material becomes separate binary columns for brick, wood, plastic, etc. Normal permutation importance shuffles each column separately, which can create impossible

combinations like "brick=yes, wood=yes, plastic=yes" for the same building. This breaks a basic rule that each building can only have one outer wall material.

We fix this by shuffling the whole categorical assignment instead of individual columns. Instead of shuffling "brick=yes/no" separately from "wood=yes/no", we shuffle which wall material each building gets assigned while keeping the one-hot format valid. Essentially a group-wise permutation importance. This is based on the ideas from Raschka (2014).

Algorithm 5: Group-wise Permutation Feature Importance

Input: Trained model $\mathcal{M}$, test data (X, y), feature groups $\mathcal{G}$
Output: Importance scores $I[g]$ for each feature group $g \in \mathcal{G}$

baseline_CI $\leftarrow$ C-index$\big(\mathcal{M}, X, y\big)$;
foreach *group* g *in* $\mathcal{G}$ **do**
  $X_{\text{shuffled}} \leftarrow X$;
  categories $\leftarrow$ onehot_to_cat$\big(X, g\big)$;
  shuffled_cats $\leftarrow$ shuffle(categories);
  $X_{\text{shuffled}} \leftarrow$ cat_to_onehot$\big(X_{\text{shuffled}}$, shuffled_cats$\big)$;
  new_CI $\leftarrow$ C-index$\big(\mathcal{M}, X_{\text{shuffled}}, y\big)$;
  $I[g] \leftarrow$ baseline_CI $-$ new_CI;
return I

To make sure our importance scores are reliable, we test them across 5 different splits using stratified cross-validation. Note we use a fixed configuration of hyperparameters based of the majority vote of parameters which we analyze in Section 8, which results in a forest of 1000 trees with a minimum 10 subjects required to split a node, and $\frac{1}{3}$ of features considered at each split. The rest of the forest is set by the default parameters of the `sksurv` implementation (Pölsterl, 2020a). The forest is trained on 1% of the dataset. Ideally we would have used the full dataset with nested cross-validation to optimize hyperparameters specifically for feature importance analysis, but the computational demands of training multiple Random Survival Forests on larger datasets made this approach impractical. This limitation means our feature importance results should be interpreted as indicative patterns rather than definitive rankings.

Figure 23 shows feature importance across all five data splits. The results give us some ideas about some patterns in the model. `Footprint` and `Outer Wall Material` consistently rank as the most important features, though which one ranks first varies between splits. `Footprint` importance ranges from about 0.057 to 0.081 across splits, while `Outer Wall Material` shows similar but more stability in scores.

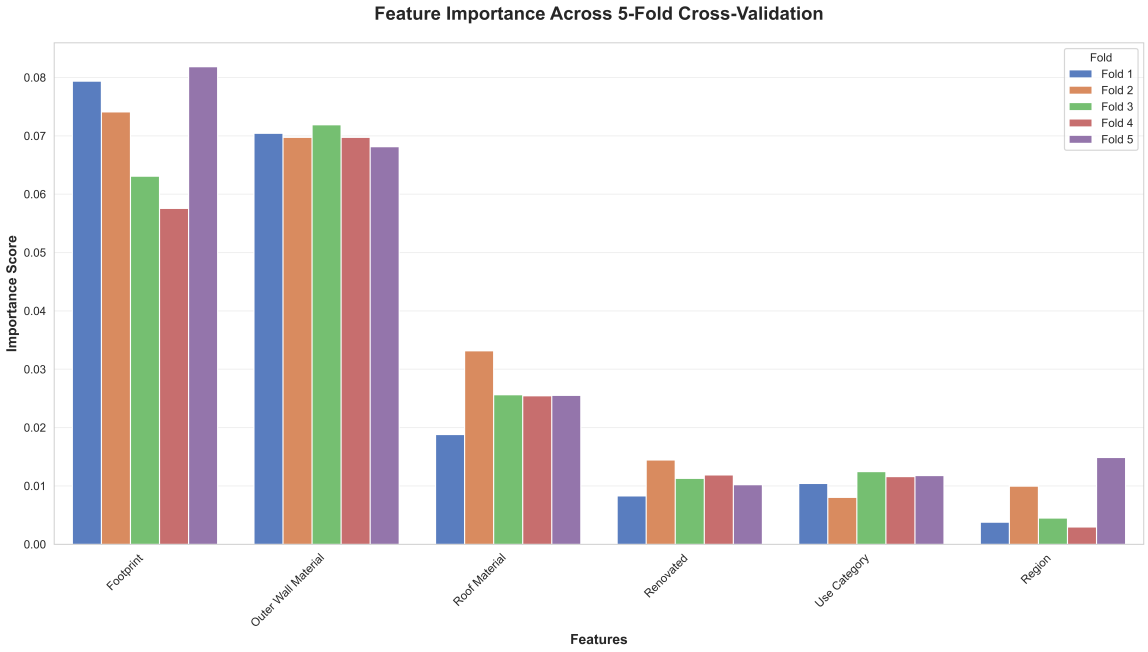

Figure 23: Feature importance scores across 5-fold cross-validation for Random Survival Forest

In contrast, `Renovated` (mean 0.0112±0.0020), `Use Category` (mean 0.0109±0.0016), and `Region` (mean 0.0072 ± 0.0045) show lower importance scores across all folds with near-zero values, indicating minimal contribution to the model's predictive performance.

For `Region`, we hypothesize this could occur because Denmark is a geographically uniform country. The climate, weather patterns, and environmental conditions that cause building wear and tear are relatively consistent across the country, so where you place a building within Denmark doesn't significantly affect its lifetime. This is not to say there aren't other regional factors that could influence demolition rates, such as local government policies, urban development pressures, or economic conditions, but our results suggest these effects are minimal compared to physical building characteristics. For `Use Category`, we believe the low importance could come from how we grouped different building types together. Our broad categories (like "Commerce" or "Other") contain many different specific building types that might have canceling effects on building lifetime. When these opposing effects are looked at together within each group, they potentially cancel each other out.

Overall, the results suggest that for Random Survival Forest physical building features are the primary determinants of building survival.

7 Deep Learning for Survival Analysis

Deep machine learning methods offer a different approach to analyzing survival data. These methods often make more accurate predictions and are able to find more complex patterns than traditional statistical models. Deep learning models can be harder to interpret than ensemble methods such as RSF, but their high efficiency and performance with very large datasets may justify the trade off. In this paper we focus on one such deep learning method, namely DeepSurv.

7.1 DeepSurv Theoretical Background

DeepSurv is a Neural Network (NN) framework based on the Cox Propertionel Hazard model described as introduced in Section 5.2.2. Like most other NN's, DeepSurv is non-linear and therefore offers the potential to find complex patterns in the data that cannot be captured by linear models. The model was introduced in the paper Katzman et al., 2018, where it was shown to perform at parity with or better than other survival functions.

The core idea of DeepSurv is to replace the linear predictor $\mathbf{x}^\top \boldsymbol{\beta}$ in Equation 29 with the output of a neural network, $f_\theta(\mathbf{x})$, where θ is the network weights. The hazard function thus becomes:

$$h(t \mid \mathbf{x}) = h_0(t) \cdot e^{f_\theta(\mathbf{x})} \tag{35}$$

DeepSurv retains the proportional hazards assumption of the Cox model, so if we compare two individuals x_i and x_j, their hazard ratio becomes,

$$\frac{h(t \mid x_i)}{h(t \mid x_j)} = e^{f_\theta(x_i) - f_\theta(x_j)}.$$

This results in $f_\theta(x)$ being a learned risk score, which can be non-linear in x, an individuals covariates.

The Neural Network Simply put, a neutral network is a sequence of simple operations that take the input vector x_i of covariates and transforms it, layer by layer, to a single output value $f_\theta(x_i)$. The process of the neural network looks like this,

1. We start with our raw vector input x_i of covariates.
2. In the first hidden layer we compute a sum of the features in the previous layer, where each feature is multiplied with a trained weight. We then add a bias to this sum, and apply a non-linear transform. In our case we use ReLU, defined as $f(x) = \max(0, x)$. Each hidden layer has several units, and this step is done separetely for each unit, giving us a new vector of number z^1.

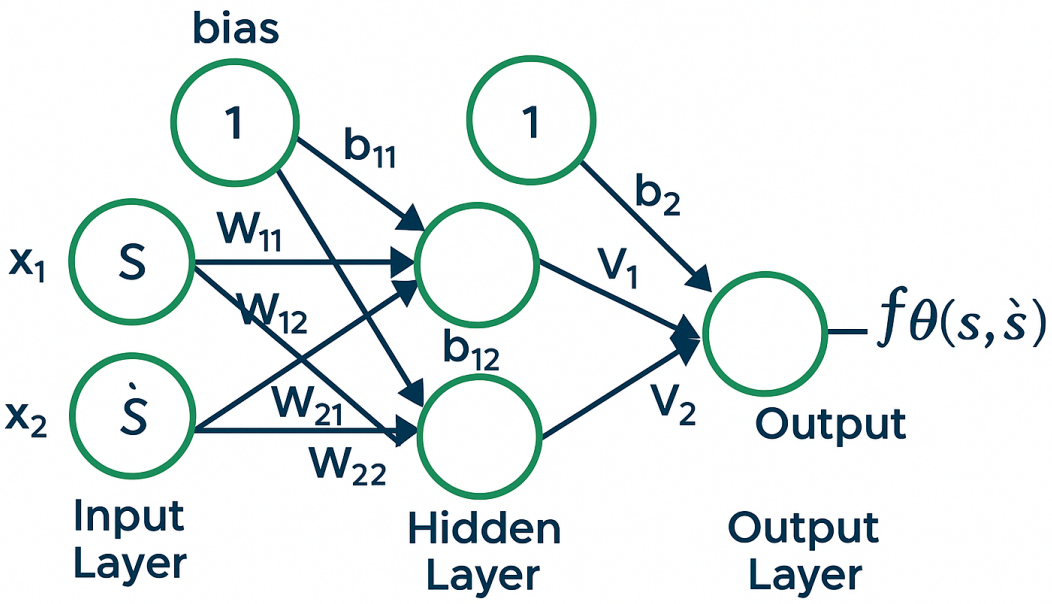

Figure 24: A neural network with one hidden layer. The input layer consists of two nodes, $x_1 = s$ and $x_2 = \dot{s}$, and a bias unit. The hidden layer includes two neurons and a bias, with weights W_{ij} and biases b_{11} and b_{12}. The output layer combines the hidden activations using weights V_1 and V_2 and bias b_2 to produce the final output $f_\theta(s, \dot{s})$. Rectified Linear Units (ReLU) are used as the activation function at all nodes. Image: GeeksforGeeks (2025).

3. In the next hidden layer we then take a weighed sum of the entries in in z^1, add a bias, apply a non-linear transform and produce z^2. This step in continued for the total number of hidden layers in the neural network. If there is only one hidden layer, this step is skipped entirely.

4. Finally, you take a weighted sum of the last hidden layers output, add a bias and apply a transform and produce a single output number $f_\theta(x_i)$.

This process is visualized in Figure 24.

Each weights and each bias is stored in the variable θ. Since we apply non-linear transforms, the final $f_\theta(x)$ can capture interactions and non-linear effects that a simple linear combination $x^T\beta$ in the Cox model cannot.

The goal when training DeepSurv is, as for Cox, to find the weights θ which makes the order of demolition observed in the training data as likely as possible. This is done in a very similar way as in Cox. By adjusting what we found in Equation 32, we can see that for DeepSurv, the probability of individual i experiancing an event at age t_i, is given by,

$$\mathcal{L}_i(\theta \mid \mathbf{x_i}) = \frac{\exp(f_\theta(\mathbf{x_i}))}{\sum_{j \in \mathcal{R}(t_i)} \exp(f_\theta(\mathbf{x_j}))}, \tag{36}$$

We then take a product over all ages where an event was observed, like we do in Equation 33, and get the partial likelihood function,

$$\mathcal{L}(\theta) = \prod_{i:\delta_i=1} \frac{\exp(f_\theta(\mathbf{x_i}))}{\sum_{j \in \mathcal{R}(t_i)} \exp(f_\theta(\mathbf{x_j}))}. \tag{37}$$

As written in tha paragraph on Cox, it is numerically easier to maximize the log likelihood. We therefore take the log of the likelihood and get,

$$\log \mathcal{L}(\theta) = \sum_{i:\delta_i=1} \left[f_\theta(\mathbf{x_i}) - \log \left(\sum_{j \in \mathcal{R}(t_i)} \exp(f_\theta(\mathbf{x_j})) \right) \right].$$

Maxmizing this with respect to θ, gives us the optimal weights $\hat{\theta}$. Mathematicaly,

$$\hat{\theta} = \arg\max_\theta \sum_{i:\delta_i=1} \left[f_\theta(\mathbf{x_i}) - \log \left(\sum_{j \in \mathcal{R}(t_i)} \exp(f_\theta(\mathbf{x_j})) \right) \right].$$

In most implementations, including `sksurv`, maximizing the log likelihood is done using the Adam optimizer (Kingma and Ba, 2015).

A potential benefit is DeepSurv's ability to capture complex functional relationships between building characteristics and demolition risk scores. Cox Proportional Hazards models learn linear relationships (risk scores are weighted sums of features, $\mathbf{x}^T\boldsymbol{\beta}$). Tree-based methods (Random Survival Forest and Gradient Boosting) learn through "if-then" rules. DeepSurv's neural network can learn smooth, non-linear functions that capture complex feature interactions through multiple non-linear transformations. This flexibility could potentially find hidden patterns in the data that simpler models cannot detect.

7.2 Feature Importance for Deep Learning Methods

Just as for Cox Proportional Hazards (Section 5.2.3) and Random Survival Forest (Section 6.3), we are once again interested in analyzing the importance of each feature

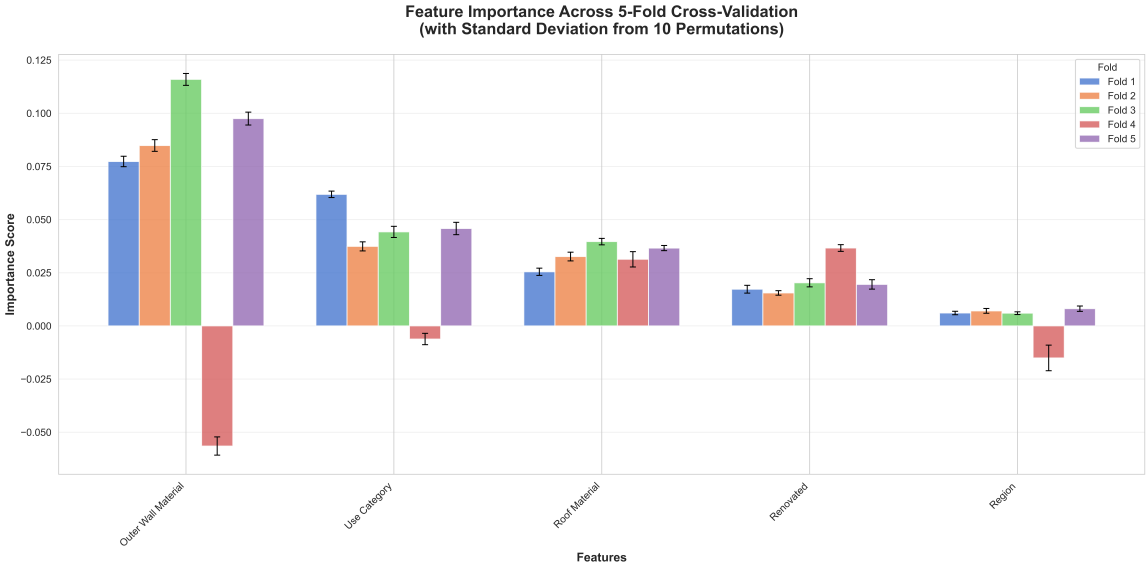

Figure 25: Feature importance scores for five different data folds, showing how each feature contributes to the model with error bars representing standard deviation from 10 permutations.

in predicting building lifetimes. We will use permutation importance following the same approach as Section 6.3, where we group the encoded features 'back' to their original category before shuffling. We again use stratified 5-fold cross-validation with 10 permutation repeats per feature group to get more robust importance estimates. This time we can train on the full dataset, but we choose to a fixed configuration for fairer comparisons to the previous analyses of feature importance. Specifically the network uses the hyperparameters specified in Section 8.3.2 – a learning rate of 0.01, a dropout rate of 0.01, and two hidden layers with 32 units each. The results of feature importance for DeepSurv is shown in Figure 25. For this analysis, we train DeepSurv on 10% of our dataset using a learning rate of 0.01, dropout rate of 0.01, and a two-layer architecture with 32 nodes per layer.

The most striking thing we notice about the plot is the odd behavior for fold four. Here, `Outer Wall Material`, `Use Category`, and `Region` have negative feature importance scores. The error bars, which are ± one standard deviation calculated from the 10 repeats, are relatively narrow too, indicating some consistency. This indicates that randomly shuffling these feature groups actually improves performance. However, when we examine the baseline C-index for fold four, we find it is 0.49730, which is essentially random guessing. As we will discuss in more depth in Section 8, DeepSurv faces some inherent challenges – whether this comes from the dataset, hyperparam-

eters, the model itself, or a combination thereof is unclear. However, one way we can overcome this specific challenge for now might be to turn a blind eye to fold four and focus on the general trends observed in the remaining folds. Once again, we emphasize that these results should be analyzed with caution and conclusions should be drawn great care.

With this in mind, `Outer Wall Material` dominates as the most important feature across all folds, with importance scores ranging from approximately 0.075 to 0.115. This aligns with our RSF findings and reinforces that the material for the outer walls of a building is important for predicting demolition risk.

The second most important feature is `Use Category`, though it shows some variability across folds, which might be a sign of inconsistent pattern recognition – however, within folds the variance is low. `Roof Material` follows with more stable importance scores, while `Renovated` contributes moderately. Finally, `Region` is last with near-zero importance, indicating that geographic location has low predictive value, just as we saw for RSF.

Notably, we had to exclude `Footprint` from the plot – which is a top predictor in RSF – because it had effectively zero feature importance. This interesting difference in feature importance between models may reflect DeepSurv's sensitivity to dataset size, it might not have been able find patterns on the continous variable. Alternatively, it could be the result of the different methods handling continuous variables differently. Recall that `Footprint` is normalized by dividing by the largest recorded value of the feature. This results in the majority of the buildings having near-zero values in `Footprint`, most likely because a few buildings are orders of magnitudes bigger than the rest. With limited training data, the neural network may struggle to learn meaningful relationships with continuous variables like building area, instead relying more heavily on categorical features like materials.

Nonetheless, the ranking of feature importance partially confirms physical properties as primary predictors, though putting more emphasis on materials, since the continuous `Footprint` does not seem to matter at all.

7.3 Implementation and Computational Considerations

DeepSurv presents unique implementation and computational challenges compared to traditional and ensemble methods. This section outlines the key design decisions and practical strategies used in adapting DeepSurv to our dataset.

Software and Libraries We implement DeepSurv using `pycox`, a Python library specifically developed for survival analysis with deep learning models and built on top of `PyTorch` (Kvamme, 2020; Kvamme, Borgan, and Scheel, 2019). The library resolves tied event times by randomly ordering events that occur simultaneously, as

opposed to employing the Breslow approximation discussed in Section 5.2.2. This only affects the composition of the risk set $\mathcal{R}(t_i)$ and, given that demolition events are sparse relative to the size of the risk set, the impact of this approximation is likely negligible (Katzman et al., 2018).

Hyperparameter Optimization The performance of DeepSurv is highly sensitive to the choice of hyperparameters. Of particular importance is the network architecture: while more complex than standard scalar parameters, we treat architectural design (i.e., the number and size of hidden layers) as a hyperparameter due to its critical role in determining model capacity and generalization. Our grid search includes configurations ranging from shallow single-layer networks to deeper multilayer architectures.

In addition, the learning rate is a dominant factor influencing training stability. Small changes to the learning rate can result in either divergence or excessively slow convergence. We also tune the dropout rate, which serves as a regularization mechanism to mitigate overfitting. While dropout serves as a powerful regularization technique, excessively high dropout rates can impair learning by limiting model capacity, particularly when training data is scarce (Goodfellow, Bengio, and Courville, 2016).

Training is performed using the Adam optimizer (Kingma and Ba, 2015), which is widely used in deep learning and has been shown to perform well across a variety of problems. We used this optimizer since it is built into `pycox` (Kvamme, 2020) and due to its prevalence in related literature such as Katzman et al. (2018).

Computational Considerations One of the most prominent challenges in deploying DeepSurv is training instability. This manifests as significant performance variation across outer cross-validation folds, as discussed in Section 8. To address this, we adopt a rigorous nested cross-validation framework with an extensive hyperparameter grid search to identify configurations that generalize well across folds as laid out in Section 3.2.

To prevent overfitting, we implement early stopping with a patience of 10 epochs on validation data created through an internal 10% stratified split. The model terminates training if no improvement occurs for 10 consecutive epochs in the partial likelihood loss. This patience mechanism handles temporary performance flukes while preventing overfitting. The implementation restores the model to its best validation state when early stopping triggers.

Despite these mitigation strategies, DeepSurv remains prone to convergence issues, underscoring the need for careful optimization and model tuning when applying deep learning methods in survival analysis contexts.

8 Comparative Analysis and Discussion of Model Performance

Having established our methodological framework and implementation of different categories of survival analysis methods, we now turn to the critical question: how do these approaches actually perform when applied to the BBR dataset?

Recall from the introduction of the thesis that we aim to answer three questions in this thesis, namely which survival analysis methods work best for building data, what building characteristics are most important for predicting service life, and how BBR data can be used for model training and prediction in survival analysis. In this section, we will attempt to answer each question by applying our evaluation framework (as discussed in Section 3.1) with actual building data.

A key part of this section is seeing how sample size impacts how well our models perform and the training time required. We start by looking at a sample of 0.1% of our data – about 4,400 buildings out of 4.4 million. This allows us to explore how hyperparameter selection varies across different data splits and sample sizes, revealing the stability (or instability) of our modeling approaches under varying data conditions. We then progressively increase the sample size and finally experiment with the full dataset. As discussed in Section 3.2 we start with such a small subset of the data due to computational limitations.

As we examine these results, remember that our goal is not to declare a single "winning" method, but to understand the trade-offs between interpretability, computational requirements, and predictive performance that characterize each approach when faced with BBR data. Furthermore, given the nature of our primary evaluation metric (the C-index), rather than focusing on predicting exact lifetimes, we focus on whether our models can correctly rank buildings according to lifetime.

8.1 Small-Scale Full Methodological Comparison

Due to the mentioned computational limitations, we trained our models on a 0.1% startified subset of the BBR dataset to preserve the original ratio of demolished (uncensored) to standing (censored) buildings. The sampling results in exactly 4,435 buildings in the dataset $\mathcal{D}_{0.1\%}$ where we preserve the 94.3% censored to 5.7% uncensored. The idea behind this approach is that our models face the same fundamental challenge as they would with the full dataset. However, this results in extremely limited learning signal with only around 250 demolition events for model training, which becomes even more constrained during cross-validation where individual folds contain even fewer actual demolitions.

We trained all five survival analysis methods on this subset using our complete nested cross-validation framework with hyperparameter grid search. This generates performance metrics for inner and outer folds that allow us to compare methods.

We structure our analysis around three main areas. First, we compare overall performance using concordance index scores from the outer cross-validation folds. Second, we examine performance patterns across both outer and inner folds to understand model stability. Third, we analyze which hyperparameters appear most frequently in our grid search, showing which model settings work best for building data.

The hyperparameter grids explored during optimization are presented in Table 9. Each method trains on roughly 20 to 30 different hyperparameter combinations, so one method doesn't inherently have an advantage over another simply based on how many different combinations it can test. We could have tested many more combinations, but this would make training much slower. Thus, the grid design was chosen based on a trade-off between exploration and computation. Most hyperparameters are separated by one full order of magnitude to help us find clear patterns in performance rather than fine-tuning small differences. Cox Proportional Hazards model (CoxPH) is the exception since it only needs to tune one main parameter (the regularization strength).

One could correctly point out that the grid is not truly 'fair' across methods, especially when comparing performance. For example, Component-wise Gradient Boosting and Gradient Boosting test 1, 10, and 100 boosting iterations, while Random Survival Forest tests 1, 10, 100, and 1000 trees in the forest. This disadvantages the boosting methods, since they might also benefit from testing 1000 estimators. However, this choice is again limited by computational constraints. Training a boosting ensemble with 1000 estimators is extremely time-intensive. Preliminary experiments showed that adding 1000 estimators to the boosting methods resulted in around a 900% increase in training time compared to excluding it from the grid search.

Similar stories could be told about other hyperparameters like the number of features considered at each split and dropout rate (see Section 6.2 and Section 7.3). While optimizing these parameters individually is not time-intensive, they create more combinations when paired with computationally expensive parameters. For example, adding just one more value to the search space of a hyperparameter that currently has 2 options increases the total combinations by 50%, while adding one more value to the search space of a hyperparameter with 3 options increases combinations by 33%. This effect forces us to reduce the search space, resulting in slightly unfair evaluation across methods.

Parameter	GB	CWB	RSF
Loss Function	{coxph}	{coxph}	-
Learning Rate	$\{0.01, 0.1\}$	$\{0.01, 0.1\}$	-
Number of Estimators	$\{1, 10, 100\}$	$\{1, 10, 100\}$	$\{1, 10, 100\,1000\}$
Max Depth	$\{3, 5\}$	-	-
Max Features	$\{0.33\}$	-	$\{0.33, \texttt{sqrt}\}$
Dropput Rate	$\{0, 0.1\}$	$\{0, 0.01, 0.1\}$	-
Min Samples Split	-	-	$\{10, 100, 1000\}$
Min Samples Leaf	-	-	$\{10\}$
Number of Combinations	24	18	24

Parameter	DeepSurv	Cox PH
Learning Rate	$\{0.0001, 0.001, 0.01\}$	-
Dropout	$\{0, 0.01, 0.1\}$	-
Architecture	$\{[16], [16, 16], [16, 16, 16], [16, 16, 16, 16]\}$	-
Alpha	-	$\{0,\ 0.0001, 0.001, 0.01\,0.1\}$
Number of Combinations	36	5

Table 9: Hyperparameter grid configurations for survival models. Top: Ensemble-based methods (GB: Gradient Boosting, CWB: Component-wise Gradient Boosting, RSF: Random Survival Forest). Bottom: DeepSurv and Cox Proportional Hazards. For DeepSurv, architecture is to be udnerstood as the number of units in each hidden layer, for instance $[16, 16]$ means two hidden layers with 16 nodes each.

With our hyperparameter grids established and nested cross-validation framework in place, we now turn to the actual performance results. For model training using the full setup with the grid search shown in Table 9 on 5 outer and 5 inner folds, we will first analyze the performance across the outer folds. Training the models on the dataset containing 0.1% of the full dataset takes 3 hours on an Apple M1 Pro chip with 8 cores and 32 GB of memory. Given that boosting methods comprise a significant portion of our 3-hour training time, scaling our current computational cost by the 10-fold increase in estimators (from 100 to 1000) would result in approximately 27 hours of total training time.

8.1.1 Outer Fold Evaluation

Looking at the outer fold evaluation results in Figure 26, several key patterns emerge from our model comparison on the 0.1% data subset.

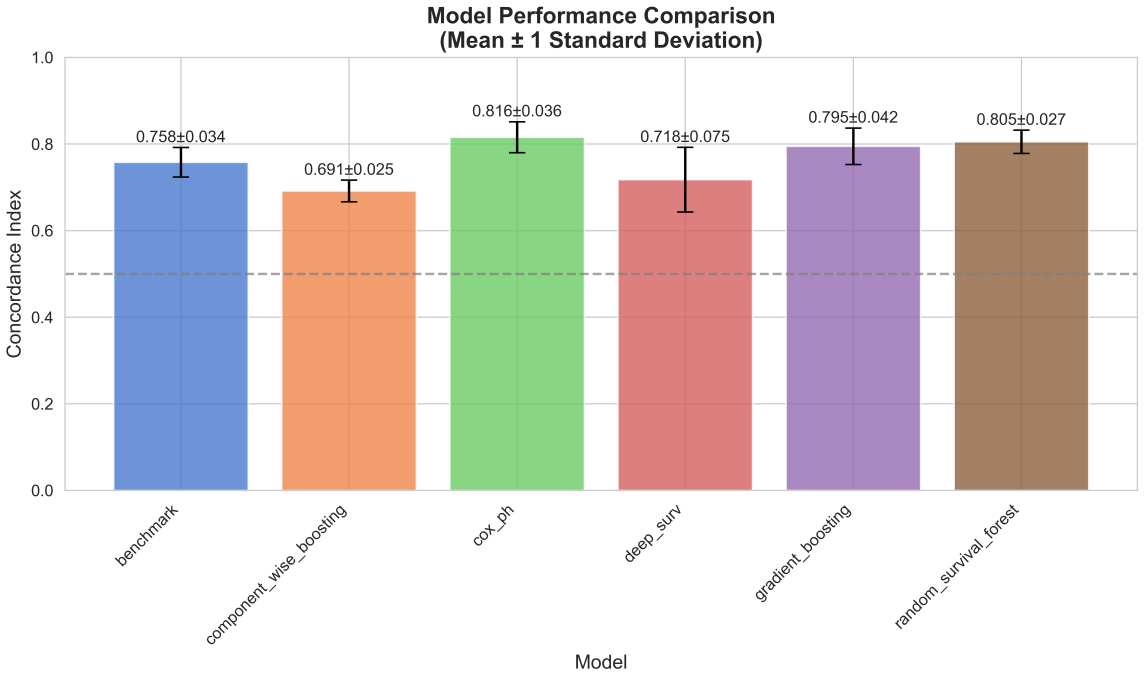

Figure 26: Performance comparison of survival models trained on 0.1% of data. Concordance index values shown as mean ± standard deviation from nested cross-validation (5 outer, 5 inner folds). Note that the standard deviation comes from only $n = 5$ C-index. Cox Proportional Hazards achieves the highest performance, followed by Random Survival Forest and Gradient Boosting.

The Cox Proportional Hazards model achieves the highest mean concordance index at 0.816 ± 0.036, showing that this traditional statistical method is good at ranking buildings by demolition risk. Random Survival Forest follows closely at 0.805 ± 0.027, while Gradient Boosting achieves 0.795 ± 0.042. The similarity in performance among these top three methods indicate that no single approach heavily outperforms the others when working with this sample size. We note that overlapping error bars do not correspond to statistical significance, and we will therefore later conduct paired t-test to evaluate if the models performance differences are statistically significant.

Our straightforward benchmark model achieves a comparatively strong performance of 0.758 ± 0.034 – even higher than DeepSurv and Component-wise Gradient

Boosting. This finding is surprising given that the benchmark model is computationally simple. Even though it can't capture complex interactions between features and requires continuous variables to be grouped into categories, it still has impressive performance.

DeepSurv achieves a more modest 0.718 ± 0.075, and Component-wise Gradient Boosting performs worst at 0.691 ± 0.025. The relatively poor performance of DeepSurv is particularly interesting given its success in healthcare applications, as discussed in the literature review. We hypothesize that this is primarily due to the sample size, especially the amount of demolitions present. On average each inner fold's test set includes only $4,435 \times 5.7\% \times 80\% \times 20\% \approx 40$ uncensored demolitions.

The fact that more complex models like DeepSurv and Component-wise Gradient Boosting cannot consistently outperform a simple benchmark model has significant implications. It might suggests that our limited dataset, particularly the small number of observed demolitions, may not provide enough information for complex models to learn the patterns and completely leverage their architectures. Thus, it might be an indication that these models require large amounts of data to effectively generalize (Bahri et al., 2024). It is also possible that they have over-fitted to the training data, leading to poorer generalization on the unseen data.

While the mean performance results provide a useful overview of model capabilities, they hide the important details about how consistently each method performs across different data subsets. The standard deviations we observed, particularly DeepSurv's notably high one, suggest that some models may be more sensitive to the specific splits of data than others. This is why we in Figure 27 show the performance trajectory of each method across the five outer folds.
DeepSurv shows unpredictable performance. In fold 4, its performance sharply dropped to 0.5907, which is only slightly better than random guessing. However, it then recovers to 0.75 in fold 5. This fluctuation explains why we saw such a high standard deviation for DeepSurv in Figure 26. It suggests that when trained on different portions of our limited dataset, the neural network struggles to identify consistent patterns.

In contrast, Cox Proportional Hazards, Random Survival Forest, and Gradient Bossting models show very stable performance. All consistently maintains concordance indices above or equal to 0.7455 across all data subsets. Overall, the similar performance trends of these three methods, all peaking in fold 2, slightly dipping in fold 3, and strongly recovering in fold 5, suggest they might be responding to similar underlying data characteristics within each subset.

Component-wise Gradient Boosting shows a very stable performance, with its concordance indices only varying between 0.6604 and 0.7230. However, this consistent stability comes with a trade-off: its performance is more or less consistently lower than the other methods. Essentially, the bias-variance tradeoff (Herlau, Schmidt,

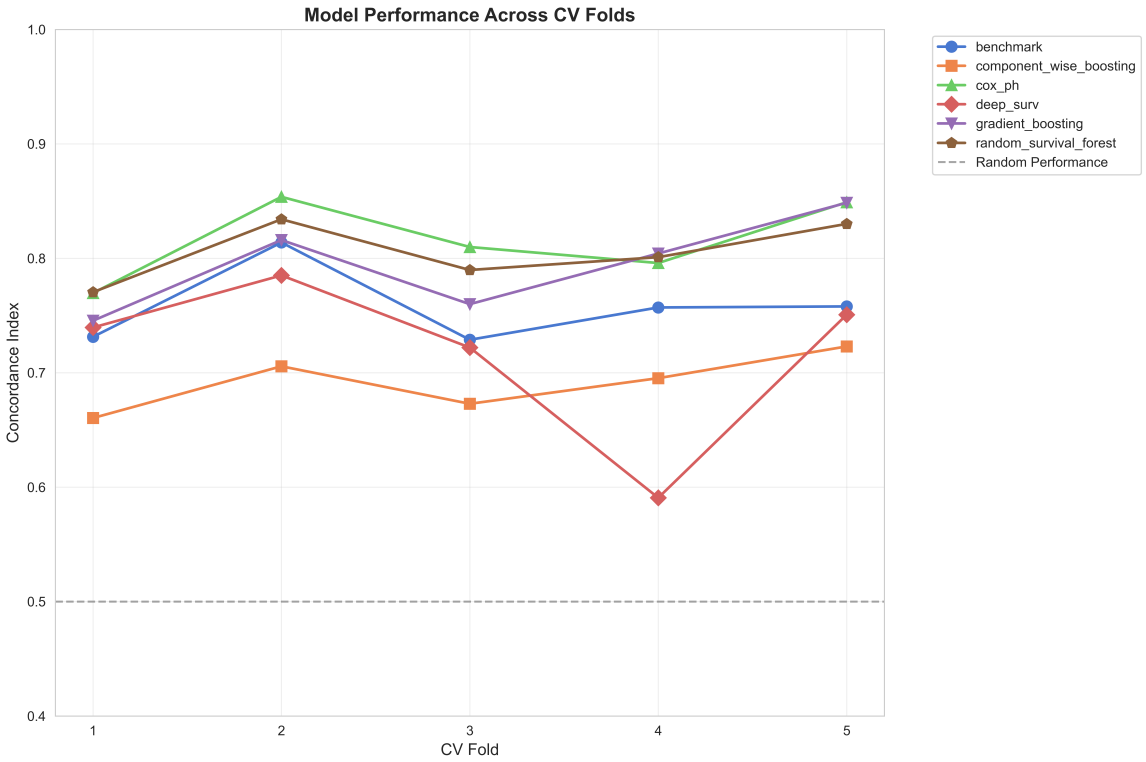

Figure 27: Cross-validation performance of survival models across individual outer folds when training on 0.1% of data.

and Mørup, 2023).

While the outer fold results show how each model performs on unseen test data, they don't tell us how effectively our hyperparameter optimization process works for each method. The inner fold scores, generated during the grid search phase of our nested cross-validation, give us insights into the hyperparameter search landscape that each model navigates through.

8.1.2 Inner Fold Evaluation

Understanding inner fold performance is interesting to examine when we are working with limited data, as it reveals whether a model's hyperparameter optimization consistently identifies good configurations, or whether it struggles to find stable solutions across different training subsets.

Figure 28 shows the distribution of inner fold scores across all hyperparameter combinations and cross-validation splits for each model. The box plots, do not only show the usual performance range during hyperparameter tuning, but it also high-

lights how consistent our search process was and points out if there exist any exceptionally low-performing configurations, which might signal bad hyperparameter choices.

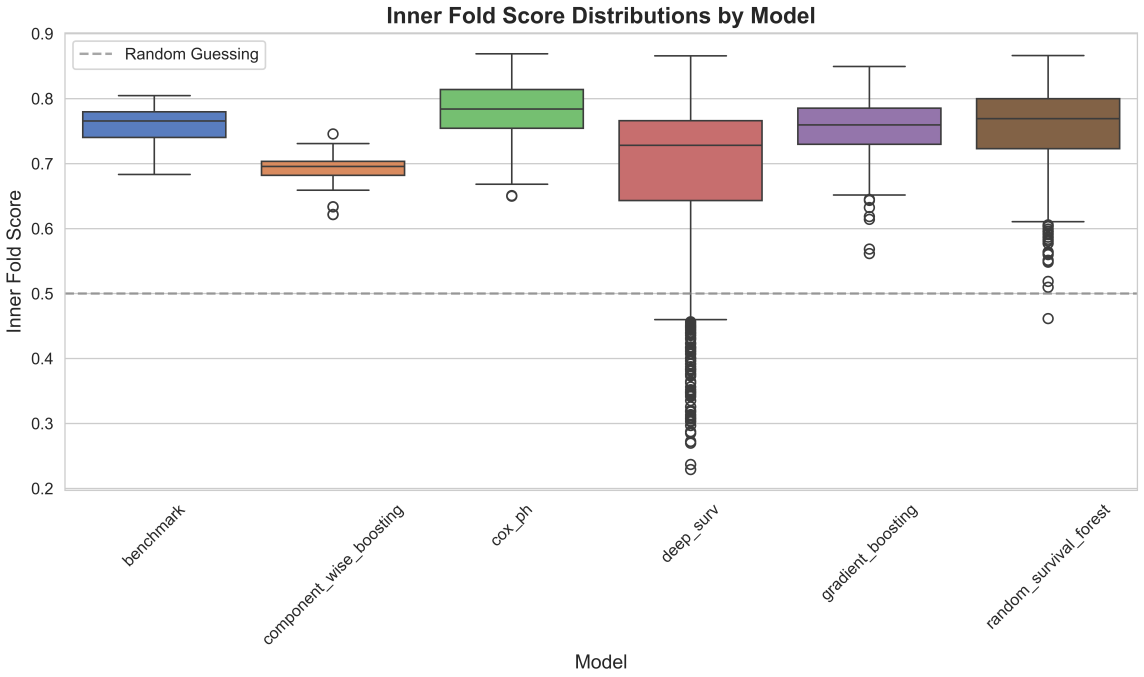

Figure 28: Distribution of inner fold performance scores (C-index) across different models. Each box plot summarizes the performance during nested cross-validation across multiple inner folds. The dashed horizontal line at 0.5 represents the performance of random guessing.

The benchmark model shows relatively stable performance, with the distribution of concordance index scores ($n = 25$) tightly clustered around the mean of 0.7586. This tight clustering is reflected in its low standard deviation (0.0304) and a narrow IQR $= 0.7799 - 0.7402 = 0.0397$.

As expected, DeepSurv's performance is very inconsistent, with C-index scores ranging from a low of 0.2291 to a high of 0.8658. This large spread is evident in its high standard deviation (0.1271) and a wide IQR $= 0.7660 - 0.6432 = 0.1228$, indicating that it's very difficult to make this model perform well consistently – especially with limited training data as discussed above.

Cox Proportional Hazards, Gradient Boosting, and Random Survival Forest perform similar in stability, in the order they appear in this sentence. This indicates these methods are less sensitive to parameter choices and work more reliably on the

sampled dataset $\mathcal{D}_{0.1\%}$.

Even though Component-wise Gradient Boosting has lower overall performance, it is the most stable of all models. Its very low standard deviation 0.0239 shows how consistently it performs, even more so than the benchmark model.

8.1.3 Selected Hyperparameters Across Cross-Validation Folds

Taking a look at which hyperparameters were chosen across different data splits within $\mathcal{D}_{0.1\%}$ helps us understand how stable each model's optimization process is. When the same parameters are selected more times, it suggests the model has clear preferences that work well with this type of data. When parameters vary widely across folds, it may indicate that the model is sensitive to small changes in the training data. Tables 10 through 19 show the hyperparameters selected for each model across all five outer cross-validation folds using Algorithm 4. Throughout the presentation of these hyperparameters, we recall that we are only looking at a 0.1% stratified sample of $\mathcal{D}$. Importantly, we should focus on whether entire hyperparameter configurations remain consistent, as machine learning models typically depend on the interaction between parameters rather than individual parameter choices in isolation.

Since the benchmark doesn't tune any hyperparameters, it is not included. Cox Proportional Hazards (Table 10) shows the most consistent choosing of hyperparameters among all methods presented in the tables. This is of course slightly biased, since it only has one hyperparameter to tune, so we do expect it to be more stable.

Parameter	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Most Common
Alpha L2 Penalty	0.001	**0.1**	**0.1**	**0.1**	**0.1**	0.1

Table 10: Selected hyperparameters by fold - Cox Proportional Hazards

After that comes Component-wise Gradient Boosting (Table 11), which chooses the same combination of hyperparameters three out of five times. The method keeps the dropout rate extremely consistent at 0, while varying a bit on the learning rate and number of estimators. A very surprising insight here, is that two out of five times, the method wants to use just one predictor. This could mean that for those specific data subsets, a single highly influential variable is enough to capture most of the predictive power, and adding more complexity by including other features does improve performance. This could also indicate that the method is being extremely conservative with the limited data, preferring to rely on the strongest signal rather than risk overfitting by adding additional, potentially weaker predictors.

Parameter	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Most Common
Dropout Rate	**0**	**0**	**0**	**0**	**0**	0
Learning Rate	**0.1**	0.01	**0.1**	**0.1**	0.01	0.1
N Estimators	**100**	1	**100**	**100**	1	100

Table 11: Selected hyperparameters by fold - Component-wise Boosting

Taking a look at the chosen hyperparameters for Random Survival Forest (Table 12), we see that it has relatively good stability. The method shows clear preferences for certain parameters, consistently choosing a max features value of 0.33 and min samples split of 1000 across all folds. The consistent choice of max features indicates that using about one-third of available features at each split provides optimal randomization without losing too much predictive power. This aligns with Hastie, Tibshirani, and J. Friedman (2009, Chapter 15).

However, it alternates between 100 and 1000 estimators, suggesting that while the model has found its sweet-spot for tree-building strategies, it adapts the ensemble size based on the data split. Some data splits may contain more consistent patterns that can be captured well with fewer trees (100), while others may have more noisy patterns requiring a larger ensemble (1000).

Parameter	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Most Common
N Estimators	**100**	10	**1000**	**1000**	**100**	100/1000 (tie)
Max Features	**0.33**	sqrt	**0.33**	**0.33**	**0.33**	0.33
Min Samples Split	**1000**	**1000**	**1000**	**1000**	**1000**	1000

Table 12: Selected hyperparameters by fold - Random Survival Forest

Table 13 shows the selected hyperparameters for DeepSurv. It shows mixed stability pattern (similar to Random Survival Forest) that once again reveals the challenges of neural network learning with limited survival data. The model shows perfect consistency in learning rate selection, always of 0.01. This preference for a "slow" learning rate makes sense given the small dataset size (Brigato and Iocchi, 2020). Higher learning rates would likely cause unstable training or poor convergence with limited examples.

However, when it comes to the regularization in terms of dropout rate, we see some uncertainty, because we have a tie between 0 and 0.1 – slightly strange, since it can also choose 0.01, but decides only to do so one out of five times. This might tell

us that sometimes the limited data doesn't provide enough signal for regularization to help, while other times it requires quite a bit of dropout (0.1).

In four out of five folds, DeepSurv chooses a simple single-layer network with 16 units, but in fold 5 it selects a deeper three-layer architecture, still with 16 units (note we always specify 16 as the number of units regardless of the depth).

Parameter	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Most Common
Learning Rate	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	0.01
Dropout Rate	**0**	**0**	**0.1**	0.01	**0.1**	0/0.1 (tie)
Architecture	**[16]**	**[16]**	**[16]**	**[16]**	[16, 16, 16]	[16]

Table 13: Selected hyperparameters by fold - DeepSurv

Finally, Table 14 shows the chosen hyperparameters for Gradient Boosting. This method shows some stability, selecting the same hyperparameter combination in two out of five folds. The model consistently chooses a max depth of 3 and 100 estimators across all folds, while having a learning rate of 0.01 selected in four out of five folds. In terms of the dropout rate, the method is torn between 0 and 0.01, as we have seen before. Note, that for this method, we tune four hyperparameters, so the fact that it chooses the same combination in two out of five folds is kind of impressive (there are 32 combinations of parameters).

The consistent selection of max depth 3 (the lowest allowed in our configuration) and 100 estimators (the highest allowed) suggests the model prefers many shallow trees over fewer deeper ones. This aligns well with the whole idea of boosting (Harvard University CS109A, 2019). However, the fact that the model consistently hits the boundaries of our search space indicates our hyperparameter ranges may be too restrictive, and thus it suggests we should expand our hyperparameter search ranges in future experiments.

Parameter	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Most Common
Dropout Rate	**0**	**0**	0.1	0.1	**0**	0
Learning Rate	0.1	**0.01**	**0.01**	**0.01**	**0.01**	0.01
Max Depth	**3**	**3**	**3**	**3**	**3**	3
N Estimators	**100**	**100**	**100**	**100**	**100**	100

Table 14: Selected hyperparameters by fold - Gradient Boosting

To summarize, the analysis of hyperparameters reveals that when predicting building demolition with limited survival data, simpler and more conservative approaches are consistently favored. Models tend to use shallow structures, slow learning, and moderate regularization. Component-wise Gradient Boosting often relies on only one predictive factor, suggesting a single strong feature can be highly influential given small data samples. In contrast, DeepSurv struggles with choosing hyperparameters, leading us to believe that deep learning methods can be less effective with limited datasets.

For ensemble methods, the number of individual predictors has proven to often play an important role in how well the overall model performs, with higher numbers of estimators generally improving performance. To understand this, we create heatmaps that show how different combinations of the number of estimators and other model hyperparameters affect performance. Each "box" represents the average performance score from the cross-validation, allowing us to see if specific combinations consistently yield better results.

These plots can in theory be shown for each combination of number of estimators with other hyperparameters – we here choose to show it for

- Random Survival Forest with number of estimators against the minimum samples in each split
- Gradient Boosting with number of estimators against the learning rate

in Figure 29 and 30. For Random Survival Forest, both parameters matter together – more trees plus more conservative splitting rules both help performance. We can hypothesize a simple intuition for this relationship: when individual estimators are kept simple through conservative splitting, an ensemble of many such estimators can capture patterns through their collective "voting power". Each tree contributes a simple prediction that aggregates into strong overall performance. On the other hand, if we use many estimators that are individually complex (through aggressive splitting), each tree becomes "too smart" for its own good, leading to overfitting. This demonstrates the fundamental ensemble principle that many weak learners often outperform fewer strong learners.

For Gradient Boosting, the heatmap reveals a different but equally intuitive pattern. The strong horizontal gradient shows that increasing the number of estimators consistently improves performance, while the learning rate has minimal impact. We can hypothesize a simple intuition here as well: gradient boosting works by sequentially adding weak learners to correct the mistakes of previous ones. With limited data, having more iterations (and thereby estimators) allows the model to step-by-step refine its predictions through many small corrections. In theory, the learning rate should significantly impact performance because it controls how aggressively the

model corrects its mistakes at each step. While we can only speculate, because it becomes practically impossible given the amount of computing power available to us, the limited impact we observe here might be explained by the amount of data in $\mathcal{D}_{0.1\%}$. So, given our setup, it could be that as long as the model has enough iterations, it can compensate for different learning speeds.

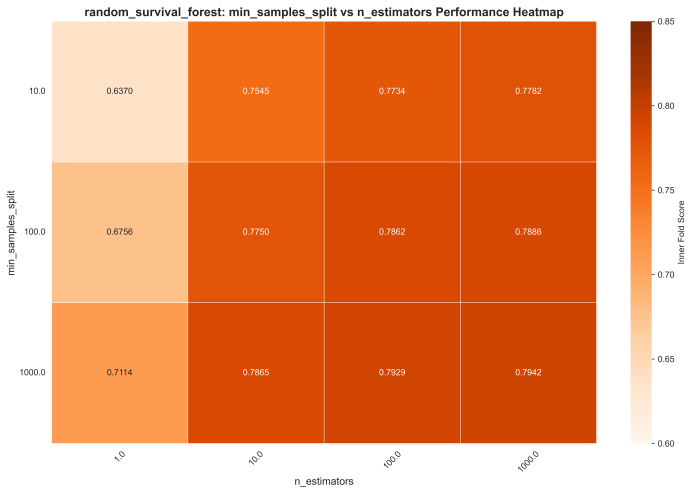

Figure 29: Random Survival Forest hyperparameter heatmap: Performance improves with both higher numbers of estimators and larger minimum samples pr. split values, with optimal results in the bottom-right region.

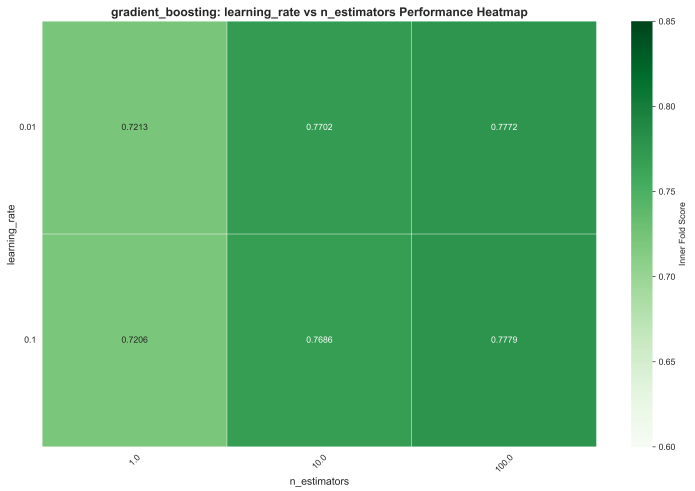

Figure 30: Gradient Boosting hyperparameter heatmap: More estimators consistently improve performance, while the learning rate has less impact on results.

8.1.4 Model Comparison

Using the model comparison framework described in Section 3.1.4, we can now examine which models truly perform better than others, rather than simply looking at average scores or scores across folds. It's important to clarify what we mean by "better" or "worse" performance in this context. We say Model A is worse than Model B in terms of C-index if there is a statistically significant difference in performance favoring Model B across cross-validation folds. Recall that the C-index does not directly measure how accurately we can predict exact building lifetimes, but rather measures each model's ability to correctly rank buildings by their demolition risk – that is, whether the model can correctly identify that Building X is more likely to be demolished before Building Y.

Table 15 presents the results of pairwise correlated t-tests that statistically compare the performance differences between all possible pairs of survival analysis models.

Each row tests whether the performance difference between two models is statistically significant, accounting for the correlation between cross-validation folds. The table provides the mean performance difference, t-statistic, p-value, and confidence interval for each comparison, allowing us to determine which model differences are statistically meaningful versus those that could be due to random variation.

Model A	**Model B**	**ΔMean**	t_{stat}	p**-value**	**CI Low**	**CI High**
CWB	RSF	-0.114	-18.125	**<0.001**	-0.131	-0.096
CWB	GB	-0.103	-9.005	**<0.001**	-0.135	-0.072
CWB	CoxPH	-0.124	-9.497	**<0.001**	-0.160	-0.088
CWB	DeepSurv	-0.026	-0.512	0.636	-0.168	0.116
CWB	Baseline	0.191	11.374	**<0.001**	0.145	0.238
CWB	Benchmark	-0.066	-3.682	**0.021**	-0.116	-0.016
RSF	GB	0.010	0.749	0.496	-0.028	0.048
RSF	CoxPH	-0.011	-1.265	0.275	-0.034	0.013
RSF	DeepSurv	0.087	1.834	0.141	-0.045	0.220
RSF	Baseline	0.305	16.831	**<0.001**	0.255	0.355
RSF	Benchmark	0.047	3.507	**0.025**	0.010	0.085
GB	CoxPH	-0.021	-1.263	0.275	-0.066	0.025
GB	DeepSurv	0.077	1.382	0.239	-0.078	0.232
GB	Baseline	0.295	10.454	**<0.001**	0.217	0.373
GB	Benchmark	0.037	1.602	0.185	-0.027	0.101
CoxPH	DeepSurv	0.098	2.237	0.089	-0.024	0.220
CoxPH	Baseline	0.316	13.168	**<0.001**	0.249	0.382
CoxPH	Benchmark	0.058	3.310	**0.030**	0.009	0.106
DeepSurv	Baseline	0.218	4.349	**0.012**	0.079	0.357
DeepSurv	Benchmark	-0.040	-0.835	0.450	-0.174	0.093
Baseline	Benchmark	-0.258	-11.237	**<0.001**	-0.322	-0.194

Table 15: Pairwise correlated t-tests between model performances. ΔMean indicates average difference in mean, with a positive ΔMean meaning Model A outperformed Model B and vice versa. Results are deemed statistically significant at $\alpha = 0.05$ if the p-value ≤ 0.05 (equivalently, the 95% confidence interval does not include zero). Significant comparisons are highlighted. Importantly, multiple comparison corrections have not been made. All tests had 4 degrees of freedom.

The results indicate clear performance tiers among the models in terms of their ranking capabilities. Component-wise Gradient Boosting (CWB) consistently demonstrates worse ranking performance compared to the other methods. It is significantly worse at ranking buildings than Random Survival Forest (RSF), Gradient Boosting (GB), Cox Proportional Hazards (CoxPH), and even our simple Benchmark model. The only model where CWB doesn't show a significant difference in ranking ability is DeepSurv. This is most likely explained by DeepSurv's performance drop in fold four, as we saw in Figure 27, where DeepSurv actually ranked buildings better than CWB in all other folds but had one tough failure. Even though CWB has poor ranking performance relative to other methods, it is important to note that it still ranks buildings significantly better than the baseline, meaning it is learning something useful about demolition risk patterns rather than just making random rankings.

Random Survival Forest shows significantly better ranking performance than our Benchmark model, which shows us that the learning capabilities of forests can provide real benefits for correctly ordering buildings by demolition risk. However, this improvement comes at a high computational cost. RSF takes much longer to train and make predictions compared to the extremely fast Benchmark model.

An interesting finding is that at face value, the three top-performing methods from Figure 27 are RSF, GB, and CoxPH. These methods however show no statistically significant differences between each other in their ranking abilities. This means we cannot confidently say that one method is better than the others at ranking buildings by demolition risk based on our data. When we say there's "no significant difference", it does not mean the models rank buildings exactly the same way, but rather that the differences we observe in their ranking performance could reasonably be due to random variation rather than one method being genuinely better at identifying demolition risk.

This finding is somewhat surprising given that CoxPH appears to rank buildings more consistently across folds when we look at Figure 27. Visually, CoxPH seems to generally have a higher C-index across most folds, but the statistical test reveals that this is not enough to be considered statistically significant.

Multiple Comparison Correction Since we compare seven methods in total – including the random benchmark (denoted "Baseline") – one could argue for using a multiple comparison correction. The reason for this is that testing multiple hypotheses increases the likelihood of Type I errors (false positives). Specifically, a significance level of $\alpha = 0.05$ implies a 5% chance of incorrectly rejecting the null hypothesis for any single test. However, when conducting multiple tests, the probability of observing at least one false positive increases according to the formula $1 - (1 - \alpha)^q$, where q is the number of pairwise comparisons. In our case, with $q = \binom{7}{2} = 21$ comparisons,

this corresponds to a $1 - 0.95^{21} \approx 66\%$ chance of encountering at least one Type I error.

To mitigate this, one common approach is the Bonferroni correction (Dunn, 1961), which adjusts the significance threshold to $\alpha_{\text{Bonferroni}} = \frac{\alpha}{q} = \frac{0.05}{21} \approx 0.0024$. Under this more conservative threshold, only comparisons with p-values below 0.0024 would be deemed statistically significant. Applying this correction to the results in Table 15, Random Survival Forest (RSF), Cox Proportional Hazards (CoxPH), and Component-wise Boosting (CWB) would no longer be significantly better than Benchmark. Likewise, DeepSurv would no longer be significantly better than Baseline.

As mentioned earlier, in this thesis we *do not* apply multiple comparison correction in our analysis which has potential drawbacks – specifically, an increased family-wise error rate of up to approximately 66%. It can be justified given that our primary interest lies in specific pairwise comparisons rather than global claims about "the best method overall," and that the research context is exploratory rather than confirmatory in its nature. As Rubin (2024) notes, for individual inferences about specific model pairs (e.g., "CoxPH is significantly better than Benchmark"), it is valid to use the standard significance level $\alpha = 0.05$ without correction, since applying Bonferroni would treat the tests as part of a family statistically while interpreting them individually. However, we acknowledge this increases the risk of Type I errors and recommend interpreting borderline significant results ($0.0024 < p < 0.05$) with additional caution.

Summary Our analysis of the 0.1% data sample included an analysis of outer fold performance, inner fold stability, and hyperparameter selection patterns, showing that Cox Proportional Hazards, Random Survival Forest, and Gradient Boosting achieved similar ranking performance, while DeepSurv showed a lot of instability and Component-wise Gradient Boosting consistently underperformed. However, we have only tested with 0.1% of our available data. We cannot be certain whether these patterns are the same with larger datasets, if models can handle more complexity when given more information, or if the performance differences between methods will remain the same. For instance, DeepSurv might close the performance gap with traditional methods when provided with sufficient data, or given more complex architecture. To examine these possibilities, we now explore how both model performance and hyperparameter preferences change across different data sizes.

Due to computational limitations, we will exclude the boosting methods (Gradient Boosting and Component-wise Gradient Boosting) from the experiments with more data. These methods need significantly longer training times, which become a problem when scaling to larger datasets, making it impractical for us to include them in our analysis of the larger data samples.

8.2 Medium-Scale Model Evaluation

The 0.1% data analysis showed that DeepSurv had unreliable performance while our simple benchmark model worked surprisingly well. To understand whether these patterns happen because the methods have limitations or because we didn't have enough training data, we test our models on a larger 5% sample, which gives us 50 times more data.

We test four methods: Cox Proportional Hazards, Random Survival Forest, DeepSurv, and our benchmark model. We skip the gradient boosting methods because they take too long to train on larger datasets with our current setup due to the reasons laid out in Section 6.2.

This larger dataset gives us 12,675 actual demolition events compared to only 253 in the smaller sample. We examine if this much larger amount of data can help DeepSurv perform more consistently, since deep learning methods usually work better when they have more examples to learn from (Zhang et al., 2020).

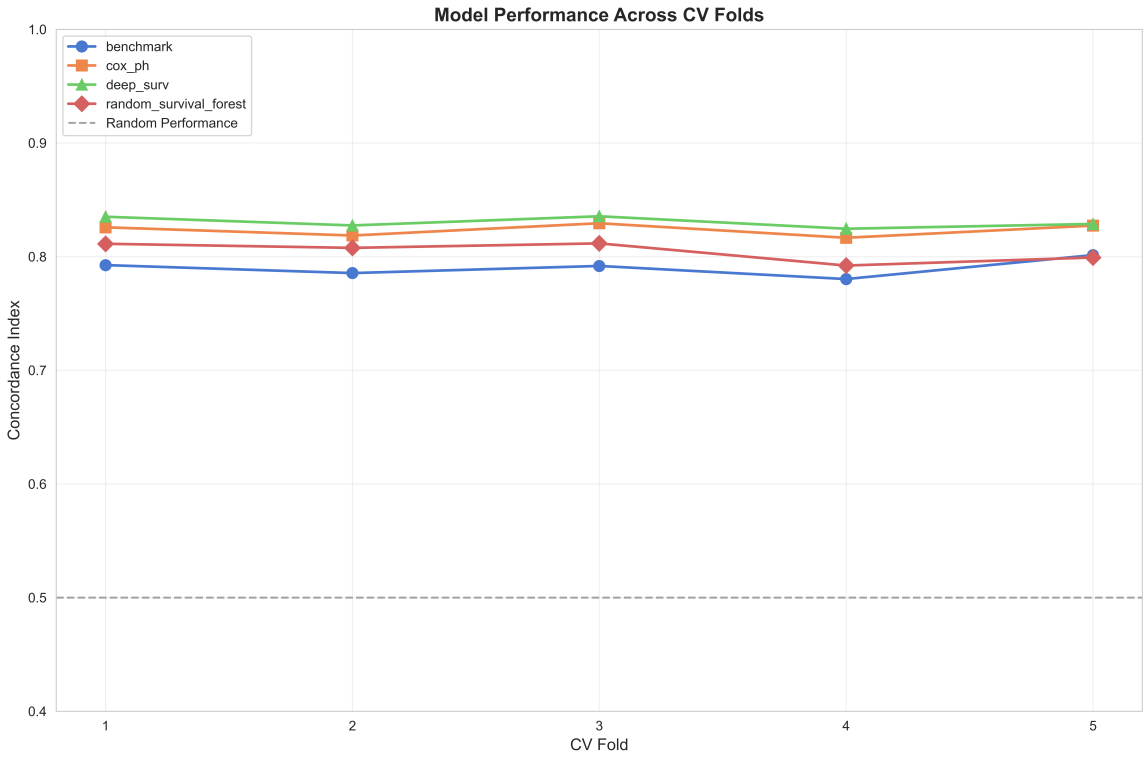

Figure 31: Cross-validation performance of survival models across individual folds when training on 5% of data. All methods show stable performance above random guessing (0.5).

The outer fold performance shown in Figure 31 shows much more stable behavior across all methods. DeepSurv now maintains consistent performance with a mean C-index of 0.8303 ± 0.0043 across all five folds. Cox Proportional Hazards and DeepSurv now achieve very similar performance levels, while Random Survival Forest follows closely with mean C-index 0.8044 ± 0.0076, and the benchmark once again maintains steady performance with 0.7903 ± 0.0071.

However, to discuss the reliability of the methods, we need once again to examine the inner folds too. For the inner fold analysis, we use violin plots instead of box plots to better visualize the density distributions of scores across all hyperparameter combinations and cross-validation splits as seen in Figure 32.

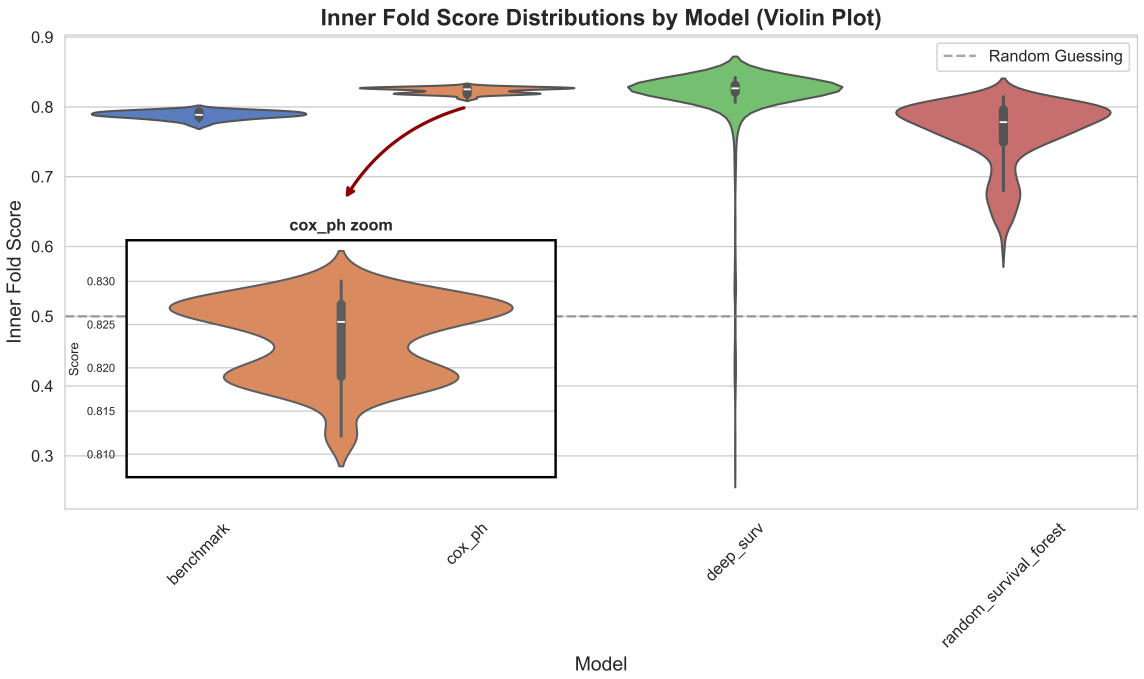

Figure 32: Inner fold score distributions for survival models trained on 5% of data. Violin plots show performance variability during hyperparameter tuning.

The Cox Proportional Hazards model has a multimodal distribution pattern, as shown in the zoomed inset, where we see that it has peaks around 0.818 and 0.827. We hypothesize that this shape means that some L2 penalty values work better while others work slightly worse, creating a "bumpy" performance landscape.

Random Survival Forest shows more spread than the benchmark and Cox methods. The distribution shows the beginning of a lower tail, suggesting some sensitivity to hyperparameter choices. It is consistent with what we see for the 0.1% data split,

so we believe the overall distribution indicates that Random Survival Forest generally performs well across most hyperparameter settings, with some but fewer configurations lead to suboptimal performance.

Even though we see improved outer fold stability, DeepSurv's inner fold distribution shows that challenges are still present. The violin plot shows a very wide distribution of C-index with a pronounced lower tail. We believe there are two possible explanations for this pattern. Firstly, certain hyperparameter combinations could systematically produce poor performance while most configurations give good results around the mean of 0.8303 ± 0.0043. Alternatively, the trained model might sometimes calculate extremely low C-index scores due to convergence issues or other problems, even with otherwise reasonable hyperparameter combinations.

To investigate these possibilities, we can examine the hyperparameter combinations that produced the 20 highest and 20 lowest C-index scores. By comparing these two groups, we can determine whether poor performance consistently comes from specific parameter combinations (supporting the first hypothesis) or whether the same hyperparameter combinations appear in both high and low-performing groups (corresponding to the second hypothesis).

Testing this, the analysis shows evidence supporting the second hypothesis of computational instability. 4 complete hyperparameter combinations appear in both the best-performing group (C-index: $[0.837 - 0.842]$) and the worst-performing group (C-index: $[0.285 - 0.564]$). That is 20% of the time.

This pattern shows that DeepSurv's poor performance is likely not caused by bad hyperparameter choices, but instead by problems during training or difficulty in reaching a stable solution. The same hyperparameter combination can lead to either good results or very bad ones, which means the training process is unstable, even when more data is available.

We dig a bit deeper, and find that 89 data points (9.89% of all DeepSurv evaluations) are beyond $1.5 \times$ IQR on the poor performance side – while these may not necessarily constitute outliers by strict statistical standards, we refer to these points as potential outliers for convenience. Even though the same parameter combinations appear multiple times both in the potential outliers and high-performing cases, there might still be some systematic patterns. Notably, 77.5% of potential outliers used the lowest learning rate (0.0001). This could suggest that very slow learning may contribute to training difficulties. The most problematic combination (appearing 10 times in the potential outliers) uses high dropout (0.1), very slow learning (0.0001), and the deepest architecture ($[16, 16, 16, 16]$). This could be an indication that the combination of aggressive regularization, minimal learning steps, and complex architecture may create unstable training conditions.

We hypothesize an explanation for this inconsistent performance is that the gradient descent optimization gets trapped in different local minima depending on random

weight initialization. When the neural network starts training, the weights are set to random values, which means each training run begins from a different point in the optimization landscape. From these different starting points, the optimization process can converge to entirely different solutions. Some good, some bad. When the learning rate is very small, the optimization algorithm takes tiny steps during training, making it difficult to "escape" from poor local minima once trapped. Essentially, the model gets stuck in a suboptimal solution and lacks the momentum to find better solution.

Additionally, it could also be that with such slow learning rates, the model may not improve within the patience window, causing early stopping to terminate training before the model has had sufficient time to learn meaningful patterns. Recall that the patience window is a fixed 10-epochs for our implementation of DeepSurv. Future work could explore dynamic patience that scales with learning rate, or treat patience as a tunable hyperparameter rather than a fixed value.

We want to stress that even though the performance for DeepSurv looks stable in Figure 31, we should interpret with care. Our extensive experience with DeepSurv throughout this thesis has revealed reliability issues with the model – from dramatic performance drops in individual folds to unpredictable sensitivity to hyperparameter choices. The inner fold analysis further supports the training instability, and this apparent success (on the outer folds) may represent a lucky coincidence rather than genuine robustness. These good outer fold results could be misleading given the model has demonstrated unreliability across our broader experimental framework.

Essentially, we must be careful about drawing strong conclusions from these patterns. The fact that identical hyperparameter combinations can give us both great and terrible results suggests that some factors during training play a large role in determining performance.

While the 5% analysis gives us some insights into model stability, we should ask ourselves an important question: would even larger datasets and more complex architectures help fix DeepSurv's training instabilities and allow us to better understand the true performance potential of these methods? To analyze this, we extend our analysis to 10% and 100% of the available data and extend the grid search.

8.3 Large-Scale Model Evaluation

Due to computational limitations, we focus on the methods: Cox Proportional Hazards, DeepSurv, and our benchmark model. The ensemble methods are excluded since they represent the primary computational bottleneck, as explained in Section 6.2. Furthermore, we believe we have seen close to the capabilities of Random Survival Forest, making it less critical to include in these larger-scale experiments.

In addition to training the models on larger datasets than in the previous section we also expand the grid search for hyperparameter tuning. This is done since the hyperparameter combinations originally chosen for the grid search were chosen for training on $\mathcal{D}_{0.1\%}$ and we know that when data is limited, any Machine Learning model is highly prone to overfit the training dataset, especially if its complexity is too high for the current task (Brigato and Iocchi, 2020). We thus introduce more complex model architectures in the grid search now that we are training on datasets of 50-100 times the size of $\mathcal{D}_{0.1\%}$. The new grid search for the DeepSurv model is shown in Table 16.

Parameter	**DeepSurv**
Learning Rate	$\{0.001, 0.01\}$
Dropout	$\{0.01, 0.1, 0.2, 0.3\}$
Architecture	$\{[16], [16, 16], [16, 16, 16],$ $[32], [32, 32], [32, 32, 32],$ $[64], [64, 64], [64, 64, 64]\}$

Table 16: Hyperparameter grid configuration for the DeepSurv model. The total of 72 combinations arises from 2 learning rates, 4 dropout values, and 9 network architectures.

In addition to allowing DeepSurv to use more complex architectures (with 32 and 64 units), the updated hyperparameter grid also includes higher dropout rates of 0.2 and 0.3. This adjustment was motivated by our observations in Section 8.1.3, where DeepSurv sometimes selected the highest available dropout rate of 0.1, suggesting that even stronger regularization may be beneficial.

8.3.1 Large-Sample Model Evaluation

We are now considering a 10% stratified subset, $\mathcal{D}_{10\%}$, which consists of 443,626 observations of which 25,352 (5.7%) have observed demolitions. We evaluate the three models – Benchmark, Cox Proportional Hazard, and DeepSurv – using the nested cross-validation framework with hyperparameter grid search explained in earlier sections. The C-index scores obtained for the models in the five outer folds are shown in Table 17 and plotted in Figure 33.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean ± Std
DeepSurv	0.8308	0.8309	0.8304	0.8318	0.4451	0.7538 ± 0.1543
Cox PH	0.8221	0.8222	0.8182	0.8228	0.8250	0.8221 ± 0.0022
Benchmark	0.7874	0.7762	0.7832	0.7770	0.7858	0.7819 ± 0.0046

Table 17: Outer fold C-index scores for each model across five folds. The DeepSurv model shows high variance due to a performance drop in one fold.

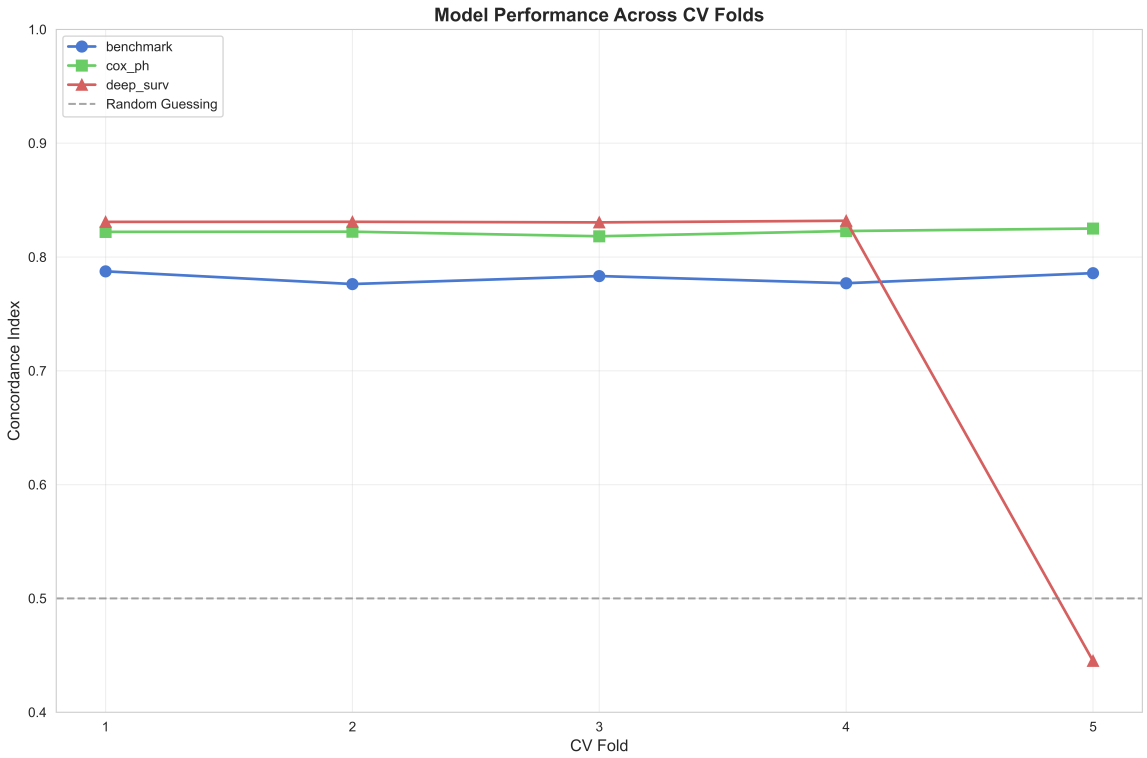

Figure 33: Plot of the outer fold C-index for each model on the 10% dataset sample. The Benchmark, Cox Proportional Hazards (Cox PH), and DeepSurv models are compared. The dashed grey line at 0.5 represents random guessing.

The results generally seem consistent across the folds for each model and follow the trend one might intuitively expect with DeepSurv outperforming CoxPH which in turn outperforms the Benchmark, but there is one very noticeable anomaly, namely DeepSurv's fifth fold. In outer fold 5 DeepSurv's C-index drops to 0.4451 after having hovered around ≈ 0.83 in the other folds. This markedly different C-index in fold 5 is

also what explains DeepSurv's comparatively high standard deviation of 0.1543. We will investigate the instability of DeepSurv more later in this section when we also look into the performance of the inner folds.

To assess whether differences in model performance are statistically significant (using C-index as the performance metric) we perform a correlated t-test (as discussed in Section 3.1.4). The result of this test are presented in Table 18.

Model A	**Model B**	**ΔMean**	t_{stat}	p**-value**	**CI Low**	**CI High**
CoxPH	DeepSurv	0.0682	0.584	0.591	-0.2562	0.3927
CoxPH	Benchmark	0.0401	10.735	**<0.001**	0.0298	0.0505
CoxPH	Baseline	0.3221	195.776	**<0.001**	0.3175	0.3266
DeepSurv	Benchmark	-0.0281	-0.240	0.822	-0.3537	0.2975
DeepSurv	Baseline	0.2538	2.193	0.093	-0.0676	0.5752
Benchmark	Baseline	0.2819	82.525	**<0.001**	0.2724	0.2914

Table 18: Pairwise correlated t-tests between model performances. ΔMean indicates average difference in mean, with a positive ΔMean meaning Model A outperformed Model B and vice versa. Results are deemed statistically significant at $\alpha = 0.05$ if the p-value ≤ 0.05 (equivalently, the 95% confidence interval does not include zero). Significant comparisons are highlighted. Importantly, multiple comparison corrections have not been made. All tests had 4 degrees of freedom.

When looking at the mean differences (denoted ΔMean) Cox Proportional Hazard model (CoxPH) seems to be the best performing model. Considering the performance of DeepSurv in outer fold 5, this might not be such a surprising discovery, even though one might initially have expected DeepSurv to have superior performance. It is important to note that CoxPH is not significantly better than DeepSurv at $\alpha = 0.05$ and that DeepSurv in fact does perform better on outer fold 1-4. Nevertheless, we will investigate this result in more depth to see if we can gauge why the performance of DeepSurv seems so unstable. On Figure 34 the C-index scores of the inner folds for all three models are shown as violin plots.

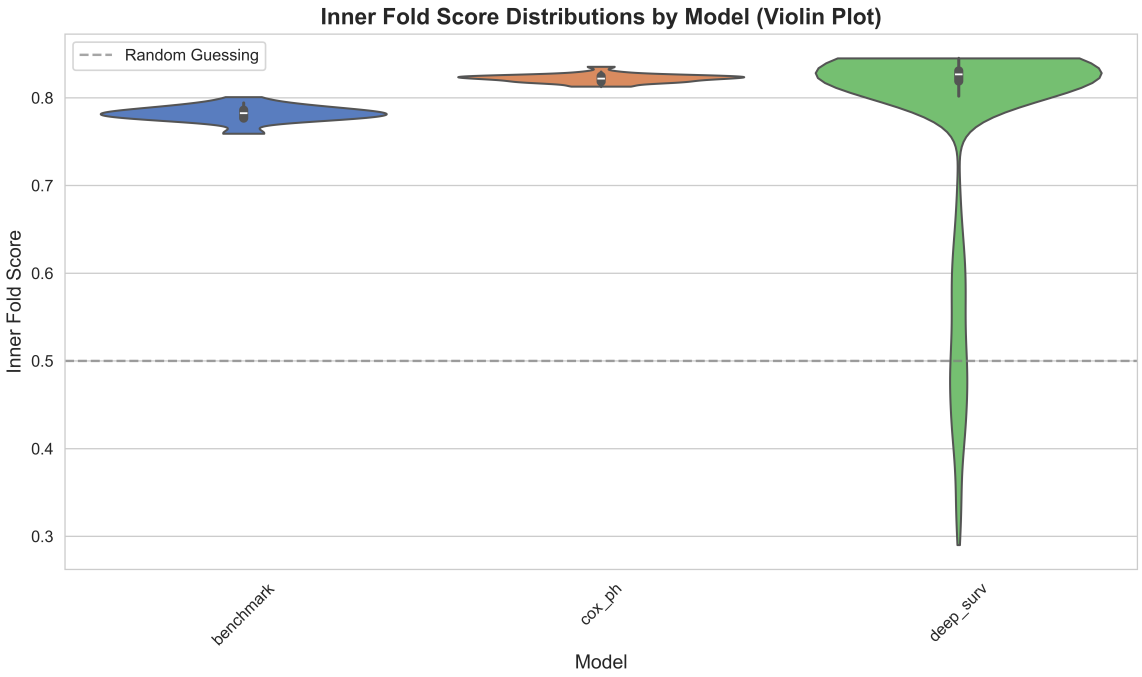

Figure 34: Violin plots of the inner fold results from training on $\mathcal{D}_{10\%}$

The violin plots show that while the Benchmark and Cox Proportional Hazards (CoxPH) models yield tightly clustered and stable inner fold performances, DeepSurv exhibits a much wider distribution with substantial variance. This could of course be the result of poor parameter configurations leading to degraded performance. However, this seems unlikely to be the full explanation. If bad performance were solely due to poor hyperparameter combinations, we would expect the outer fold results to be relatively insulated from this effect – as each outer fold is evaluated using the best-performing hyperparameters identified through inner cross-validation. The fact that outer fold 5 performs dramatically worse than the others, despite using the same grid search framework and a reasonably large training set (443,626 observations), suggests that the model's training process may be unstable. This points toward the possibility of training stochasticity (e.g., weight initialization, convergence behavior), local minima, or data-specific characteristics in fold 5 that make generalization particularly difficult for DeepSurv.

To better understand what might have caused the model failure in outer fold 5, we now examine the specific hyperparameters selected for DeepSurv in each fold. While nested cross-validation aims to mitigate overfitting to hyperparameters, it is still possible for the search to converge on suboptimal configurations. The table below shows the chosen hyperparameters and resulting performance scores for each outer

fold, along with the most commonly selected hyperparameters across folds.

Parameter	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Most Common
Dropout Rate	**0.01**	0.1	**0.01**	**0.01**	0.1	0.01
Learning Rate	**0.01**	0.001	**0.01**	**0.01**	**0.01**	0.01
Architecture	**[32, 32]**	**[32, 32]**	[64, 64]	**[32, 32]**	[64, 64]	[32, 32]
Score	0.8308	0.8309	0.8304	0.8318	0.4451	0.7538 ± 0.1543

Table 19: Selected hyperparameters and performance — DeepSurv

Looking at the selected hyperparameters provides a potential explanation. Table 17 shows that fold 5 is the only fold to select the deeper architecture `[64, 64]` with a relatively high dropout rate of 0.1. In contrast, the other outer folds mostly use the shallower $[32, 32]$ architecture with dropout 0.01. While this difference might seem small, it reflects a general pattern we have observed throughout the thesis: DeepSurv is highly sensitive to its hyperparameters. At both the 0.1% and 5% data levels, we saw how different configurations could lead to large variations in performance. The unexpectedly poor performance in outer fold 5 suggests that this sensitivity persists even with much larger training data. Although nested cross-validation is meant to guard against overfitting to hyperparameters, our coarse grid and limited compute resources may have allowed a suboptimal configuration to slip through. This highlights a key challenge when working with flexible models like DeepSurv: even when using rigorous validation procedures, unstable results can still occur.

To further investigate DeepSurv's potential at full scale, we also trained the model on all five outer folds using the most frequently selected hyperparameter configuration from the 10% experiment. The results are presented in the following subsection.

8.3.2 Full-Sample Model Evaluation

Using this fixed configuration, we now evaluate DeepSurv on the full dataset, $\mathcal{D}_{100\%}$. This allows us to assess whether its performance improves and stabilizes when trained on more data, or whether the instability observed earlier persists. The results across the five outer folds are shown in Table 20 and visualized in Figure 35, with statistical comparisons presented in Table 21.

Model	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Mean ± Std
DeepSurv	0.8352	0.8351	0.5461	0.6289	0.8321	0.7355 ± 0.1236
Cox PH	0.8223	0.8232	0.8209	0.8226	0.8225	0.8223 ± 0.0008
Benchmark	0.7861	0.7859	0.7847	0.7848	0.7832	0.7850 ± 0.0010

Table 20: Model performance across five outer folds. Reported values are Concordance Index (C-index) per fold and the overall mean ± standard deviation.

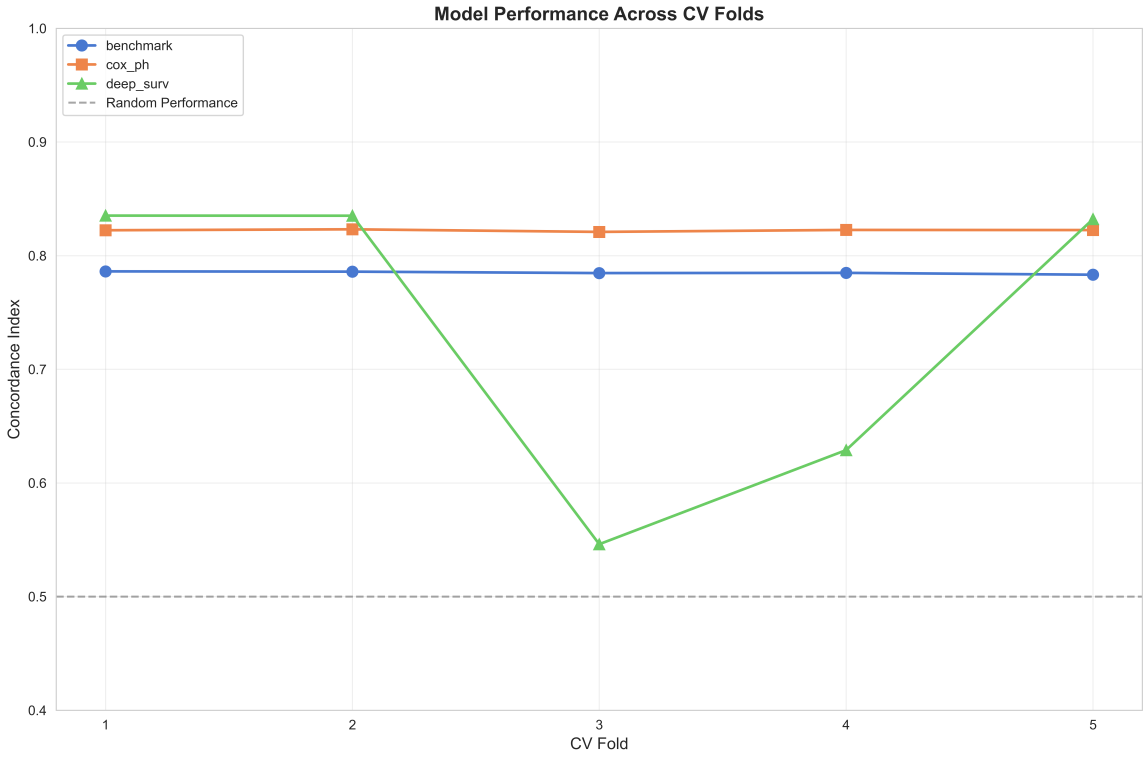

Figure 35: Plot of the outer fold C-index for each model.

The results on $\mathcal{D}_{100\%}$ echo the trends observed at the 10% level. While DeepSurv performs well in folds 1, 2, and 5, it suffers substantial performance drops in folds 3 and 4, with C-index scores of 0.5461 and 0.6289, respectively. This leads to a high standard deviation of 0.1236 — not due to natural variability, but as a result of unstable behavior across folds. CoxPH, by contrast, maintains consistent and reliable performance, with nearly identical C-index scores in all five folds and a mean

of 0.8223. The Benchmark model again performs consistently but remains clearly inferior to the more advanced models.

Model A	Model B	ΔMean	t_{stat}	p-value	CI Low	CI High
CoxPH	DeepSurv	0.087	0.940	0.400	-0.169	0.343
CoxPH	Benchmark	0.037	43.432	**<0.001**	0.035	0.040
CoxPH	Baseline	0.322	566.425	**<0.001**	0.321	0.324
DeepSurv	Benchmark	-0.049	-0.534	0.621	-0.307	0.208
DeepSurv	Baseline	0.235	2.539	0.064	-0.022	0.493
Benchmark	Baseline	0.285	363.317	**<0.001**	0.283	0.287

Table 21: Pairwise correlated t-tests between model performances. ΔMean indicates average difference in mean, with a positive ΔMean meaning Model A outperformed Model B and vice versa. Results are deemed statistically significant at $\alpha = 0.05$ if the p-value ≤ 0.05 (equivalently, the 95% confidence interval does not include zero). Significant comparisons are highlighted. Importantly, multiple comparison corrections have not been made. All tests had 4 degrees of freedom.

The statistical comparisons in Table 21 reinforce these patterns. CoxPH significantly outperforms the Benchmark, while the difference between CoxPH and DeepSurv remains statistically insignificant ($p = 0.400$), as does the difference between DeepSurv and Benchmark ($p = 0.621$). These results suggest that while DeepSurv has the capacity for high performance, its sensitivity to training dynamics or fold-specific characteristics undermines its reliability. This highlights a key challenge in applying deep learning models like DeepSurv: even with access to a lot of data, performance instability can persist.

8.4 Summary of Model Comparison Results

We do not declare a single "winning" model – nor was that our objective. Instead, our results show that each method has its own pros and cons. Model selection should be based on problem specific requirements such as interpretability, computational constraints, robustness, and model complexity. That said, our findings suggest that ensemble methods, despite their computational cost, offer no clear performance advantage over simpler models like Cox Proportional Hazards. Given this imbalance, ensemble methods appear less promising for survival analysis in this context.

Our most extensive comparison was done using 0.1% of the dataset ($\mathcal{D}_{0.1\%}$). At this scale, Cox Proportional Hazards (CoxPH), Random Survival Forests (RSF), and

Gradient Boosting (GB) models were the strongest performers. Surprisingly, our simple Benchmark model also outperformed some of the more complex alternatives. DeepSurv performed poorly and showed substantial instability, while Component-wise Gradient Boosting (CWB) was weak but consistent. Even at this small data scale we encountered long training and prediction times for the ensemble methods. CoxPH proved stable and fast with minimal hyperparameter tuning.

As we increased the amount of data to $\mathcal{D}_{5\%}$, we were forced to drop both gradient boosting methods due to computational constraints. RSF maintained its performance level, suggesting that data scarcity was not its primary limitation. DeepSurv showed apparent improvements in outer fold performance at this scale, but closer inspection revealed inconsistencies in the inner folds. CoxPH and the Benchmark model continued to deliver stable results.

At $\mathcal{D}_{10\%}$, we excluded all ensemble models due to their significant computational demand. We thus continued our investigation with DeepSurv and CoxPH, including deeper architectures and updated hyperparameters for DeepSurv. Despite these adjustments, DeepSurv continued to perform inconsistently. While the majority of outer folds showed improved performance, we still saw instability in both inner and outer folds. In contrast, CoxPH and the Benchmark model again demonstrated consistent results.

We gave DeepSurv one final opportunity by training it on the full dataset (approximately 4.3 million buildings) using the most chosen hyperparameter configuration from the $\mathcal{D}_{10\%}$ run. Even under these conditions, DeepSurv failed to achieve consistent results. Meanwhile, CoxPH and the Benchmark model remained stable and reliable.

In summary, DeepSurv shows the highest performance potential, but its instability limits its potential in its current form. Ensemble methods, particularly RSF and GB, were too resource-intensive to evaluate at larger data scales, leaving open the possibility that they may perform better with more computing power available. Nevertheless, given their steep computational demands and lack of clear superiority at small scale, their practical utility appears limited.

CoxPH stands out as the most robust and scalable method in our analysis. It consistently performed well, required little tuning, and scaled efficiently to larger datasets. The Benchmark model shares many of these traits and may have further potential with additional tuning.

We emphasize that all findings should be interpreted in light of two significant limitations. First, most experiments were conducted on subsets of the full dataset due to computational limits, and results may not generalize to larger-scale runs. Second, the dataset itself exhibits survivorship bias and contains inconsistencies in data quality. Such issues may inflate or deflate model performance. Future studies using higher-quality or more representative data could yield different results.

Finally, model interpretability varies across the tested methods. Cox Proportional Hazards model is the most interpretable – its hazard ratios directly show risk, like the 10.99 times higher demolition risk for plastic walls versus brick (Section 5.2.3).

Random Survival Forest and DeepSurv falls in the middle, since we can see which feature groups matter most through importance scores, but cannot explain it for each one-hot encoded feature (i.e. we can see that materials are important, but not which materials). An interesting finding is that these methods disagree on certain features: RSF ranks `Footprint` very highly with a mean importance of 0.71, whereas DeepSurv gives `Footprint` effectively zero importance. However, both methods agree that physical building properties are highly important, with DeepSurv weighing them slightly higher on average (excluding the anomalous results from fold 4 discussed in Section 7.2). While it certainly is possible to look deeper into tree splits or network weights to understand these models better, we did not do this analysis. Future research could investigate how to better explain what these complex models are actually learning in the context of Danish building data.

9 Future Research

The research presented in this thesis opens several promising directions for future exploration, especially since our work serves as a proof-of-concept for the evaluated methods. In this section we introduce some of the areas that could offer interesting opportunities for later research.

Methodological Enhancements One natural extension of the work presented on machine learning models would be to combine it with the work presented in Droob and Nybroe, 2024. One could use their work to estimate a baseline hazard h_0, which could be used to make predictions on lifetimes with DeepSurv or CoxPH.

As discussed in Section 3.1.1, there are other performance metrics which could be interesting to use to evalute the models. For instance, Brier Score and Integrated Brier Score could be used to evaluate both the calibration and accuracy of lifetime predictions of the models.

In addition to DeepSurv, the library `pycox` offers other Deep Learning methods suited for survival analysis, such as MTLR (Multi-Task Logistic Regression), LogisticHazard, and DeepHit. We have not looked into these, but they could be worth investigating.

The Benchmark model introduced in Section 5.2.1 presents several opportunities for enhancement. Feature weighting could be implemented, such as weighting by variance to prioritize features with greater predictive power. The impact of different binning strategies for continuous variables could also be investigated, or even

alternative approaches for including continuous variables all together. Furthermore, addressing multicollinearity by grouping highly correlated features and computing Kaplan-Meier estimates for these groups could reduce the risk of double-counting. Such improvements, potentially optimized through cross-validation, might transform the Benchmark from a baseline comparison tool into a competitive prediction model.

Data Quality Improvements Significant improvements could be made through better collaboration with both BBR and Rigsarkivet to obtain higher-quality historical data. A deeper investigation into demolition indicators and the underlying data structure would benefit from direct collaboration with BBR. Understanding how BBR intended their data to be used would reduce assumptions about the data and enable more appropriate analytical approaches. This collaboration could extend to making the BBR system more accessible and logically organized for research purposes.

In addition, the BBR dataset contains several unexplored features that could enhance model performance, particularly for deep learning approaches. Analysis of these additional features represents a significant opportunity for improvement. Furthermore, comparing results with building datasets from other countries could provide valuable insights into the generalizability of the findings and reveal country-specific patterns in building lifetimes.

Practical Applications and Impact Future research could also focus on translating analytical findings into practical applications, for instance for CO_2 emissions reduction. This could include quantifying differences in carbon emissions between different building materials (brick, wood, etc.) and identifying material constraints for specific use cases. Understanding how to effectively implement research findings in construction practice represents a crucial step toward achieving meaningful environmental impact through improved building lifetime predictions.

10 Conclusion

The primary objectives of this thesis were to (1) investigate the applicability of different survival analysis methods to Danish building data, (2) explore how certain building features impact the lifetime predictions of the models, and (3) figure out how Danish building data can be used for survival analysis.

Our investigations involved extensive data preparation, methodological development, and systematic model comparison. We cleaned and merged the BBR dataset with an auxiliary source, handling significant data quality challenges including inconsistent demolition recording and system changes over time.

We derived and implemented several survival analysis approaches: traditional methods (Kaplan-Meier, Nelson-Aalen, Cox Proportional Hazards), a Benchmark model, ensemble methods (Random Survival Forest, Gradient Boosting, and Component-wise Gradient Boosting), and deep learning (DeepSurv). Using nested cross-validation with stratified sampling, we trained and evaluated these methods across different data scales, from small subsets of 0.1% to the complete dataset. We measured performance through Harrell's concordance index that quantifies a model's ability to correctly rank buildings by lifetime.

How can Danish building data be used in survival analysis? Danish building data can be used in survival analysis by treating each building as an observation and modeling its time from construction to demolition as a time-to-event outcome. This is possible because the dataset provides relevant features as well as information from which event indicators can be extracted. However, this was only possible because of thorough cleaning and preprocessing – without it, the data would have been unsuitable for survival analysis. The raw BBR data contained $6,087,324$ buildings, and required filtering. The cleaned dataset resulting from this (containing $4,436,278$ buildings) is used in an exploratory data analysis, where we discover a need for further preprocessing. We identified buildings with implausible demolition patterns that we hypothesize resulted from administrative system changes rather than actual demolitions. Following this analysis, we created a final one-hot encoded dataset for model training, $\mathcal{D}$, containing $4,307,475$ individual buildings. Although the dataset contains right-censored observations, survival analysis methods can handle such censoring.

Which building characteristics matter most? Cox Proportional Hazards model provided direct feature importance through model coefficients, quantifying each one-hot encoded feature's individual effect relative to a reference feature. Random Survival Forest and DeepSurv required permutation importance analysis, which measures how much performance drops when one feature is randomly shuffled while the rest are kept the same. For this approach, each building can only have a single one-hot encoded feature within each feature (e.g., brick within outer wall material)

be equal to one. Thus, we measured importance at the feature "group" level rather than for individual one-hot encoded features to maintain valid constraints.

Physical building properties consistently appeared among the strongest predictors of demolition risk across all methods where feature importance analysis was conducted. Outer wall material, roof material, and footprint ranked highest in feature importance for Random Survival Forest, while building use category was more important for DeepSurv. Specifically, for Random Survival Forest outer wall material and footprint showed mean feature importance of 0.069 and 0.071, respectively. For DeepSurv (excluding fold 4 in the analysis), the outer wall material had mean importance of 0.093, while the footprint was effectively disregarded by the model. DeepSurv assigned the use category a mean feature importance of 0.047 compared to Random Survival Forest's substantially lower score of 0.011.

Cox Proportional Hazards model showed similar results, with outer wall materials having notable effects on demolition risk. Using brick as the reference category, plastic cladding and glass panels showed hazard ratios of 10.99 and 7.16 respectively, indicating increased demolition risk compared to brick. On the other hand, timber framing showed a protective effect with a hazard ratio of 0.64. For roof materials (relative to felt roofing), clay tiles demonstrated reduced demolition risk (hazard ratio 0.51). These effects must be interpreted cautiously as they may reflect confounding factors rather than causal relationships.

Geographic location within Denmark showed minimal predictive value across all methods. Importantly, all these findings must be interpreted carefully due to survivorship bias.

Which survival analysis methods show most potential? Performance varied across data scales, with no single method consistently showing superior performance. At the 0.1% subset ($\mathcal{D}_{0.1\%}$), Cox Proportional Hazards achieved the highest C-index of 0.816 ± 0.046, followed by Random Survival Forest (0.805 ± 0.027) and Gradient Boosting (0.795 ± 0.042). Our simple Benchmark model achieved 0.758 ± 0.034, often scoring higher than more sophisticated approaches like DeepSurv (0.718 ± 0.075) and Component-wise Gradient Boosting (0.691 ± 0.025).

As we increased the amount of data to $\mathcal{D}_{5\%}$, DeepSurv improved dramatically to 0.8303 ± 0.0043, while Cox Proportional Hazards maintained stable performance around 0.8235 ± 0.0050 and Random Survival Forest achieved 0.8044 ± 0.0076. However, this apparent stability for DeepSurv at the $\mathcal{D}_{5\%}$ subset may have been coincidental rather than indicative of genuine robustness. At all scales, DeepSurv suffered from severe training instability. At $\mathcal{D}_{10\%}$, it dropped to 0.7538 ± 0.1543 due to complete failure in one fold (C-index of 0.4451), and at the full dataset achieved only 0.7355 ± 0.1236 with failures in multiple folds.

Cox Proportional Hazards demonstrated the most consistent performance across all scales, and also has minimal computational requirements as well as interpretable

results. The Benchmark model, which combines feature-specific Kaplan-Meier estimators, maintained consistent performance (0.7850 ± 0.0010 on full data) with minimal computational needs. Random Survival Forest and Gradient Boosting achieved comparable predictive performance to Cox Proportional Hazards model at small scales but required more computing power and training time. Component-wise Gradient Boosting showed stable but consistently lower performance (0.691 ± 0.025 at $\mathcal{D}_{0.1\%}$) across all evaluations. Although DeepSurv showed the highest performance potential in individual cases, the method suffered from training instability, making it unsuitable for reliable practical application in its current form.

Several important limitations must be considered when interpreting our findings. Computational constraints limited most experiments to small data subsets, potentially not allowing all methods to achieve their full modeling capabilities. Furthermore, the left-truncation of the dataset very likely results in survivorship bias, potentially affecting both feature importance analysis and ranking capabilities. Finally, the findings may not generalize beyond Denmark due to different construction practices, climates, etc. in other countries.

These findings represent a proof of concept, creating a foundation for future research within survival analysis of Danish building data. Future investigations could focus on improving data quality, addressing the stability issues observed with the deep learning approach DeepSurv, and exploring additional feature engineering techniques to better capture the complex factors influencing building survival.

A Appendix

A.1 Stratified K-fold Example

Taken with respect to Algorithm 1 Consider a dataset with number of folds $K = 3$:

- Event samples ($E = 1$): indices $\mathcal{I}_1 = \{10, 23, 45, 56, 72\}$
- Censored samples ($E = 0$): indices $\mathcal{I}_0 = \{11, 14, 29, 34, 51, 63, 80\}$

After independent shuffling:

- Shuffled event indices: $\mathcal{I}_1^{\text{shuffled}} = [45, 10, 72, 56, 23]$
- Shuffled censored indices: $\mathcal{I}_0^{\text{shuffled}} = [51, 34, 14, 80, 29, 11, 63]$

The stratified assignment proceeds as follows:

For event samples ($E = 1$):

$$\begin{aligned}
\text{index}_1 = 45 &\mapsto \text{fold}_{((1 \bmod 3)+1)} = \text{fold}_2,\\
\text{index}_2 = 10 &\mapsto \text{fold}_{((2 \bmod 3)+1)} = \text{fold}_3,\\
\text{index}_3 = 72 &\mapsto \text{fold}_{((3 \bmod 3)+1)} = \text{fold}_1,\\
\text{index}_4 = 56 &\mapsto \text{fold}_{((4 \bmod 3)+1)} = \text{fold}_2,\\
\text{index}_5 = 23 &\mapsto \text{fold}_{((5 \bmod 3)+1)} = \text{fold}_3.
\end{aligned}$$

For censored samples ($E = 0$):

$$\begin{aligned}
\text{index}_1 = 51 &\mapsto \text{fold}_{((1 \bmod 3)+1)} = \text{fold}_2,\\
\text{index}_2 = 34 &\mapsto \text{fold}_{((2 \bmod 3)+1)} = \text{fold}_3,\\
\text{index}_3 = 14 &\mapsto \text{fold}_{((3 \bmod 3)+1)} = \text{fold}_1,\\
\text{index}_4 = 80 &\mapsto \text{fold}_{((4 \bmod 3)+1)} = \text{fold}_2,\\
\text{index}_5 = 29 &\mapsto \text{fold}_{((5 \bmod 3)+1)} = \text{fold}_3,\\
\text{index}_6 = 11 &\mapsto \text{fold}_{((6 \bmod 3)+1)} = \text{fold}_1,\\
\text{index}_7 = 63 &\mapsto \text{fold}_{((7 \bmod 3)+1)} = \text{fold}_2.
\end{aligned}$$

Resulting stratified folds:

$$\begin{aligned}
\text{Fold}_1 &= \{72,\ 14,\ 3\} && \big(1 \text{ event, } 2 \text{ censored}\big),\\
\text{Fold}_2 &= \{45,\ 56,\ 51,\ 80,\ 63\} && \big(2 \text{ events, } 3 \text{ censored}\big),\\
\text{Fold}_3 &= \{10,\ 23,\ 34,\ 29\} && \big(2 \text{ events, } 2 \text{ censored}\big).
\end{aligned}$$

Note that each fold maintains roughly the same event-to-censored ratio as in the original dataset (5:7 $\approx$ 0.714), with fold ratios of $1:2$ (0.5), $2:3$ (0.667), and $2:2$ (1.0). With a larger dataset, the ratio will be closer to the same.

A.2 Nested Stratified K-fold Visualization

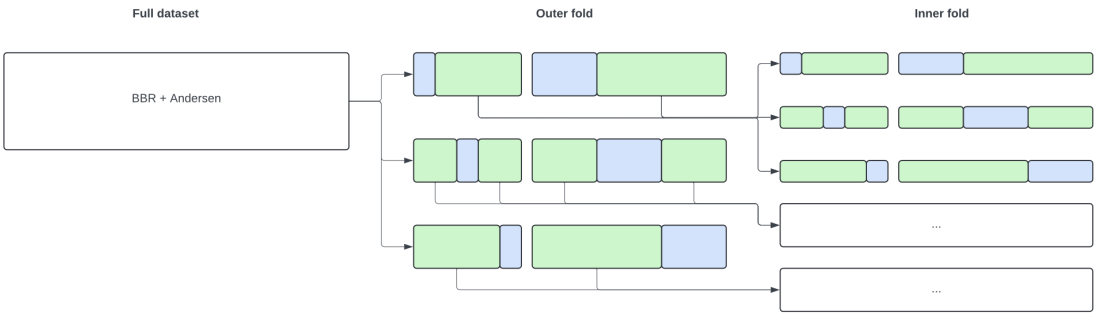

Figure 36: Nested Stratified K-Fold Cross Validation

A.3 BBR Feature Translations

- forretningsproces: Business Process
- id_namespace: ID Namespace
- byg134QualityOfCoordinateSet: Quality of Coordinate Set
- virkningFra: Effect From
- byg021BygningensAnvendelse: Use Category
- kommunekode: Municipality Code
- byg026Opførelsesår: Year of Construction
- byg404Koordinat: Coordinate
- Omtilbygningsår: Year of Renovation or Extension

A.4 Random Survival Forest Algorithm

Algorithm 6: Random Survival Forest

Input: Training data D, number of trees B, number of candidate attributes m at each split, minimum number of unique events in a terminal node d_0

Output: Ensemble Cumulative Hazard Function (CHF)

for $b = 1$ **to** B **do**

- Draw a bootstrap sample of observations D_b from the original data D;
- Initialize a survival tree T_b;
- **while** *nodes can be further split* **do**
  - At current node, randomly select p candidate attributes from all available attributes;
  - **for** *each candidate attribute* **do**
    - Identify potential split points;
    - **for** *each potential split point* **do**
      - Calculate the log-rank test statistic;
    - Select the split point with the highest log-rank statistic;
  - Split the node using the attribute and split point with the highest overall log-rank statistic;
  - Stop splitting if a terminal node has fewer than d_0 unique events;
- For each terminal node in T_b, calculate the node-specific CHF;

Calculate the ensemble CHF by averaging the CHFs across all trees: $\hat{H}_{\text{ensemble}}(t|X) = \frac{1}{B}\sum_{b=1}^{B}\hat{H}_b(t|X)$;

return $\hat{H}_{ensemble}(t|X)$

Note: This algorithm can also be used to predict the survival function using the Kaplan-Meier estimator within the terminal nodes. However, when using the `predict` method within `sksurv` for calculating the concordance index, the Nelson-Aalen estimator is used to calculate the ensemble Cumulative Hazard Function.

A.5 Demolition Rates for Outdated Categories

Code	Description	Demolition Rate
210	Agricultural/industrial production	92.7%
290	Other agri/industrial	90.8%
230	Utility building (electricity, gas, water)	90.7%
440	Daycare institution	90.4%
220	Industrial production	90.1%
320	Office, retail, storage	89.5%
420	Education/research	86.1%
330	Hotel/service business	85.5%
310	Transport/garage facility	84.3%
530	Sports facility	83.0%
390	Other commercial/transport	82.7%
410	Cultural/public building	70.7%
520	Holiday home, hostel (not summer house)	74.2%
430	Health facility	68.2%
490	Other institution (military, prison)	47.6%
130	Row or duplex house	1.9%
Roof Material Code		
80	None	74.7%

Table 22: Demolition rates for discontinued building use and roof material categories.

A.6 Full Stacked Histogram of Construction Years by Building Usage

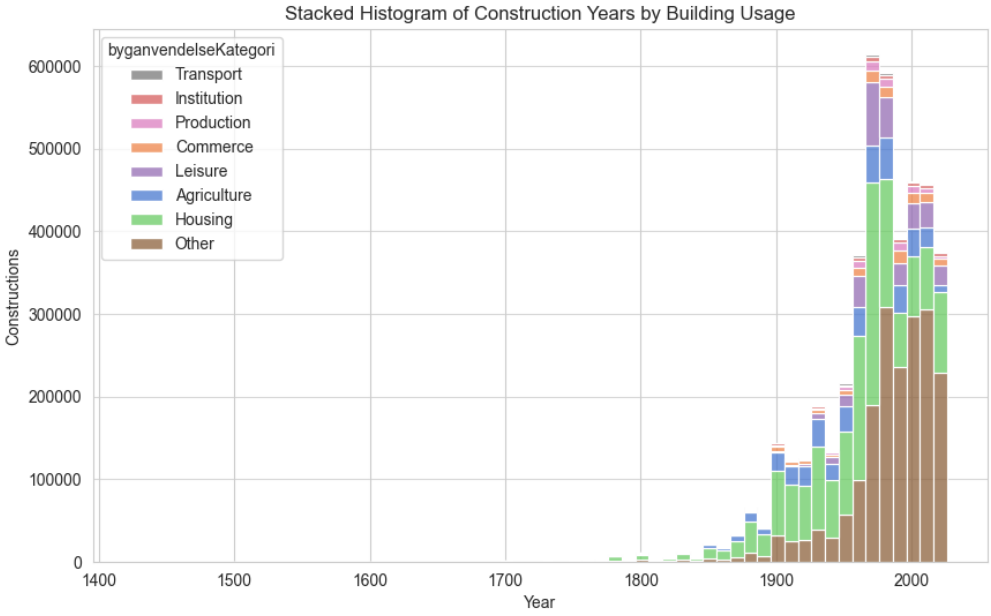

Figure 37: Building construction by usage category from 1426 to 2025. The stacked histogram shows the evolution of construction activity across eight building types.

A.7 Violin Plot Building Lifetime by Region

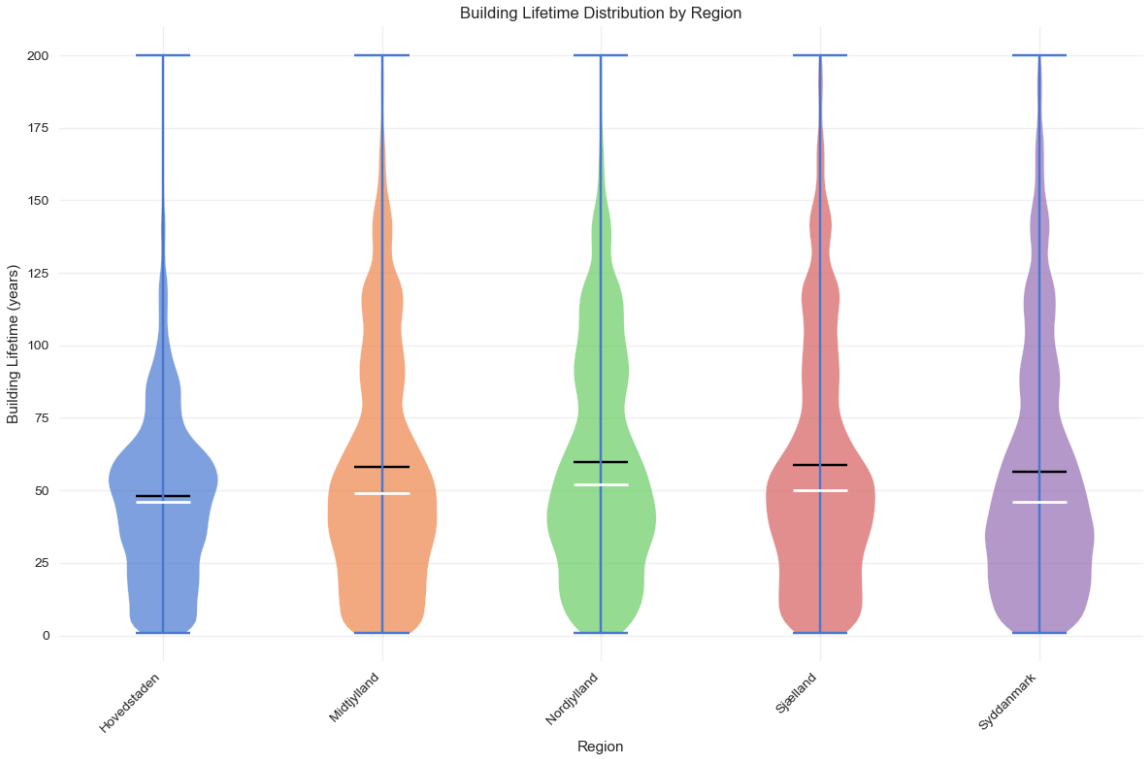

Figure 38: Building lifetime distributions across Danish regions show consistent patterns, with most structures lasting 25-75 years regardless of location. Violin plots display probability density distributions with median (white lines) and mean (black lines) values for each region.RetryClaude can make mistakes. Please double-check responses.

A.8 Buildings Demolished vs Constructed

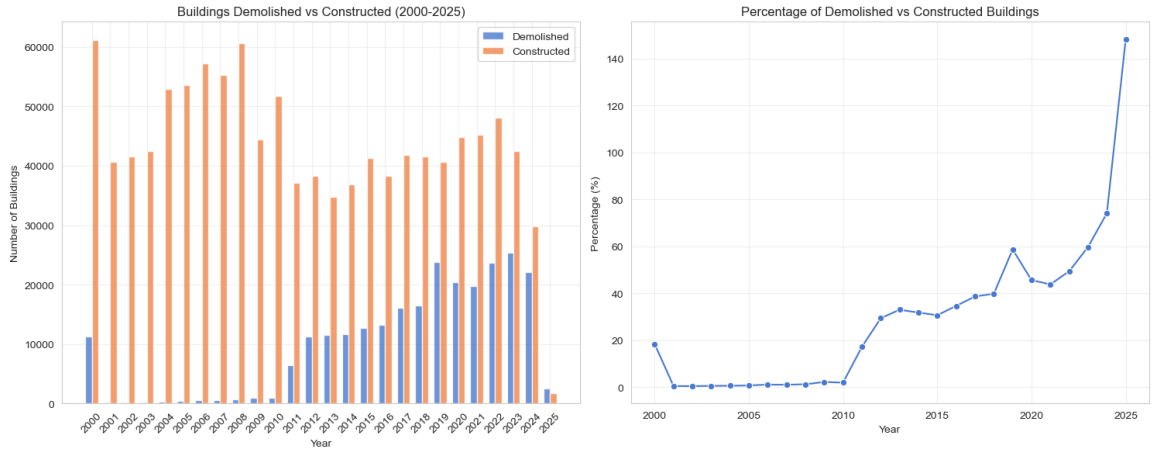

Figure 39: Comparison of building demolitions versus new construction in Denmark (2000-2025). The left panel shows absolute numbers of demolished (blue) and constructed (orange) buildings by year. The right panel displays the percentage ratio of demolitions to new construction, highlighting the dramatic increase in this ratio after 2013.

A.9 Translation Key From Number to Material

Wall Material Codes	Meanings
1	Brick
2	Lightweight concrete blocks
3	Fiber cement including asbestos
4	Timber framing
5	Wood
6	Concrete elements
8	Metal
10	Fiber cement without asbestos
11	Plastic materials
12	Glass
80	None
90	Other material

Figure 40: Wall Material Codes

Roof Material Codes	Meanings
1	Roofing felt with low slope
2	Roofing felt with high slope
3	Fiber cement including asbestos
4	Concrete tiles
5	Clay tiles
6	Metal roof
7	Thatch
10	Fiber cement without asbestos
11	Plastic materials
12	Glass
20	Living roofs
80	None
90	Other material

Figure 41: Roof Material Codes

A.10 Cox PH Beta Coefficients

Region	Hazard Ratio
Midtjylland	1.35
Nordjylland	1.40
Sjælland	0.93
Syddanmark	1.11

Table 23: Estimated hazard ratios by region (vs. Hovedstaden).

Use Category	Hazard Ratio
Commerce	1.34
Housing	0.49
Institution	0.99
Leisure	0.82
Other	1.26
Production	1.33
Transport	0.89

Table 24: Estimated hazard ratios by use category (vs. Agriculture).

Code	Wall Material (English)	HR
2	Aerated concrete block	2.31
3	Fiber cement (incl. asbestos)	3.47
4	Timber framing	0.64
5	Wood	3.85
6	Precast concrete panels	3.42
8	Metal cladding	6.26
10	Fiber cement (no asbestos)	6.55
11	Plastic cladding	10.99
12	Glass panels	7.16
80	None	7.70
90	Other / mixed materials	2.40

Table 25: Estimated yearly hazard ratios by exterior wall material (reference: code 1, brick).

Code	Roof Material (English)	HR
2	Felt roofing (steep slope)	0.72
3	Fiber cement (incl. asbestos)	0.72
4	Concrete roof tiles	0.66
5	Clay tile	0.51
6	Metal roofing	0.74
7	Thatch roofing	0.46
10	Fiber cement (no asbestos)	0.74
11	Plastic materials	1.53
12	Glass roofing	1.42
20	Green roof	1.95
90	Other / mixed materials	1.30

Table 26: Estimated yearly hazard ratios by roof-covering material (reference: code 1, felt roofing low slope).

Continuous / Binary Feature	Hazard Ratio
Bebygget Areal (per m^2)	0.000004
Om-tilbygget (yes vs. no)	0.49

Table 27: Estimated hazard ratios for the continuous and binary features.

A.11 Combined Cleaned Dataset

Feature	Missing %
forretningsproces	0.01
id_lokalId	0.00
kommunekode	0.00
registreringFra	0.01
registreringTil	100.00
virkningFra	0.00
virkningTil	99.94
status	0.01
byg007Bygningsnummer	0.01
byg021BygningensAnvendelse	0.00
byg024AntalLejligheder MedKøkken	100.00
byg025AntalLejligheder UdenKøkken	100.00
byg026Opførelsesår	0.00
byg027OmTilbygningsår	80.66
byg030Vandforsyning	96.80
byg031Afløbsforhold	96.54
byg032YdervæggensMateriale	0.78
byg033Tagdækningsmateriale	0.00
byg034SupplerendeYdervæggensMateriale	98.39
byg035SupplerendeTagdækningsMateriale	98.96
byg036AsbestholdigtMateriale	99.76
byg037KildeTilBygningensMaterialer	0.07
byg038SamletBygningsareal	44.08
byg039BygningensSamledeBoligAreal	58.83
byg040BygningensSamledeErhvervsAreal	83.58
byg041BebyggetAreal	0.01
byg042ArealIndbyggetGarage	98.17
byg043ArealIndbyggetCarport	97.91
byg044ArealIndbyggetUdhus	92.16
byg045ArealIndbyggetUdestueEllerLign	96.21
byg046SamletArealAfLukkedeOverdækningerPåBygningen	99.93
byg047ArealAfAffaldsrumITerrænniveau	99.94
byg048AndetAreal	99.36
byg049ArealAfOverdækketAreal	91.46
byg050ArealÅbneOverdækningerPåBygningenSamlet	99.99
byg051Adgangsareal	99.68

Feature	Missing %
byg052BeregningsprincipCarportAreal	100.00
byg053BygningsarealerKilde	0.07
byg054AntalEtager	43.59
byg055AfvigendeEtager	97.51
byg056Varmeinstallation	35.88
byg057Opvarmningsmiddel	73.42
byg058SupplerendeVarme	78.50
byg069Sikringsrumpladser	99.46
byg070Fredning	99.75
byg071BevaringsværdighedReference	100.00
byg094Revisionsdato	2.28
byg111StormrådetsOversvømmelsesSelvrisiko	99.85
byg112DatoForRegistreringFraStormrådet	99.85
byg113Byggeskadeforsikringsselskab	96.87
byg114DatoForByggeskadeforsikring	98.33
byg119Udledningstilladelse	99.70
byg121OmfattetAfByggeskadeforsikring	96.93
byg122Gyldighedsdato	95.93
byg123MedlemskabAfSpildevandsforsyning	99.99
byg124PåbudVedrSpildevandsafledning	99.98
byg125FristVedrSpildevandsafledning	99.99
byg126TilladelseTilUdtræden	100.00
byg127DatoForTilladelseTilUdtræden	100.00
byg128TilladelseTilAlternativBortskaffelseEllerAfledning	100.00
byg129DatoForTilladelseTilAlternativBortskaffelseEllerAfledning	100.00
byg130ArealAfUdvendigEfterisolering	99.91
byg131DispensationFritagelseIftKollektivVarmeforsyning	99.97
byg133KildeTilKoordinatsæt	1.21
byg135SupplerendeOplysningOmKoordinatsæt	16.06
byg136PlaceringPåSøterritorie	4.46
byg403ØvrigeBemærkningerFraStormrådet	99.96
byg404Koordinat	1.06
byg406Koordinatsystem	1.06
byg150Gulvbelægning	100.00
byg151Frihøjde	100.00
byg152ÅbenLukketKonstruktion	100.00
byg153Konstruktionsforhold	100.00
source	0.00
demolition_year	94.29

Feature	**Missing %**
byganvendelseKategori	0.02
region	0.17
290 Sagstype	99.99
StatusKode_ByggesagsNiveau	99.99
292 Byggesagskode	99.99
AnmeldelseAfByggearbejdeDato	99.99
DatoModtagAnsoegBygTillad	99.99
StatusKode_Byggesag	99.99
Ejerforholdskode_Ejendom	99.99
Ejendomsnummer_Ejendom	99.99

References

[1] N.-J. Aagaard et al. *Levetider af bygningsdele ved vurdering af bæredygtighed og totaløkonomi*. Dansk. SBI 30. SBI forlag, Nov. 2013.

[2] O. O. Aalen. "Nonparametric Inference for a Family of Counting Processes". In: *The Annals of Statistics* 6.4 (1978), pp. 701–726. DOI: 10.1214/aos/1176344247.

[3] Agency for Climate Data. *Basic Public Data*. Agency for Climate Data, Danish Ministry of Climate, Energy and Utilities. URL: https://www.eng.klimadatastyrelsen.dk/data/basic-public-data (visited on 06/01/2025).

[4] A. Altmann et al. "Permutation importance: a corrected feature importance measure". In: *Bioinformatics* 26.10 (May 2010). Epub 2010 Apr 12. PMID: 20385727, pp. 1340–1347. DOI: 10.1093/bioinformatics/btq134.

[5] Y. Amemiya and I. Yalcin. "Nonlinear Factor Analysis as a Statistical Method". In: *Statistical Science* 16.3 (Aug. 2001), pp. 275–294. DOI: 10.1214/ss/1009213729.

[6] R. Andersen and K. Negendahl. "Lifespan prediction of existing building typologies". In: *Journal of Building Engineering* 65 (2023), p. 14. ISSN: 2352-7102.

[7] Y. Bahri et al. "Explaining Neural Scaling Laws". In: *Proceedings of the National Academy of Sciences (PNAS)* 121.27 (2024), e2311878121. DOI: 10.1073/pnas.2311878121. arXiv: arXiv:2102.06701 [cs.LG]. URL: https://doi.org/10.1073/pnas.2311878121.

[8] P. Breheny. *Hazard functions*. Lecture notes from The University of Iowa. Accessed June 13, 2025. 2015. URL: https://myweb.uiowa.edu/pbreheny/7210/f15/notes/8-27.pdf.

[9] P. Breheny. *Tied survival times; estimation of survival probabilities*. Lecture notes from The University of Iowa. Accessed June 6, 2025. 2017. URL: https://myweb.uiowa.edu/pbreheny/7210/f17/notes/11-02.pdf.

[10] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[11] L. Brigato and L. Iocchi. "A Close Look at Deep Learning with Small Data". In: *Proceedings of the 25th International Conference on Pattern Recognition (ICPR)*. arXiv:2003.12843 [cs.LG]. 2020, –.

[12] P. Bühlmann and T. Hothorn. "Boosting algorithms: regularization, prediction and model fitting". In: *Statistical Science* 22.4 (2007), pp. 477–505.

[13] Bygnings- og Boligregistret (BBR). *BBR - Bygnings- og Boligregistret*. https://bbr.dk/forside. Accessed: May 22, 2025.

[14] Bygnings- og Boligregistret (BBR). *Kodelister*. https://teknik.bbr.dk/kodelister. Accessed: May 18, 2025.

[15] G. C. Cawley and N. L. C. Talbot. "On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation". In: *Proceedings of the 2010 International Conference on Data Mining*. Norwich, United Kingdom, 2010.

[16] D. R. Cox. "Regression Models and Life-Tables". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 34.2 (1972), pp. 187–220. URL: http://www.jstor.org/stable/2985181 (visited on 05/22/2025).

[17] N. A. Dam et al. "The Housing Bubble that Burst: Can House Prices be Explained? And Can Their Fluctuations be Dampened?" In: *Monetary Review* 1st Quarter (2011), pp. 47–69.

[18] Danish Ministry of Finance. *ADAM: A Model of the Danish Economy*. Tech. rep. https://www.dst.dk/da/Statistik/ADAM/Multiplikatorer/multiplikatorer. Danish Ministry of Finance, 2024.

[19] Danske Regioner. *Om de fem regioner*. Accessed: May 22, 2025. URL: https://www.regioner.dk/om-os/om-de-fem-regioner/.

[20] DR. *Tusindvis af boligejere risikerer at blive ramt af gamle fejl i 2023*. Accessed: 2025-06-09. Jan. 16, 2023. URL: https://www.dr.dk/nyheder/penge/tusindvis-af-boligejere-risikerer-blive-ramt-af-gamle-fejl-i-2023.

[21] O. S. Droob and M. F. Nybroe. "The fit of the survival: A statistical analysis of building survival rates". Bachelor's Thesis. Lyngby, Denmark: DTU, Department of Applied Mathematics and Computer Science, Dec. 2024.

[22] O. J. Dunn. "Multiple comparisons among means". In: *Journal of the American Statistical Association* 56.293 (1961), pp. 52–64. DOI: 10.1080/01621459.1961.10482090.

[23] T. R. Fleming and D. P. Harrington. *Counting Processes and Survival Analysis*. John Wiley & Sons, 1991. ISBN: 978-0-471-52218-4.

[24] N. Francart, T. Widström, and T. Malmqvist. "Influence of methodological choices on maintenance and replacement in building LCA". In: *International Journal of Life Cycle Assessment* 26 (Nov. 2021), pp. 2109–2126. DOI: 10.1007/s11367-021-01985-z. URL: https://doi.org/10.1007/s11367-021-01985-z.

[25] J. H. Friedman. "Greedy Function Approximation: A Gradient Boosting Machine". In: *Annals of Statistics* 29.5 (2001), pp. 1189–1232.

[26] GeeksforGeeks. *Neural Network*. Neural network structure visualization. 2025. URL: https://www.geeksforgeeks.org/bias-neurons-using-r/ (visited on 06/14/2025).

[27] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[28] F. E. Harrell et al. "Multivariable prognostic models: issues in developing models, evaluating assumptions and adequacy, and measuring and reducing errors". In: *Statistics in Medicine* 15.4 (1996), pp. 361–387.

[29] Harvard University CS109A. *CS109A Introduction to Data Science: Standard Section8 – Review Trees and Boosting (AdaBoost, Gradient Boosting, XGBoost)*. Lecture notes. Fall 2019 — Instructors: Pavlos Protopapas, Kevin Rader, Chris Tanner; Section Leaders: Marios Mattheakis, Abhimanyu Vasishth, Robbert Struyven. 2019. URL: https://harvard-iacs.github.io/2019-CS109A/.

[30] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second. Springer Series in Statistics. Springer New York, 2009. ISBN: 9780387848587. URL: https://books.google.dk/books?id=tVIjmNS3Ob8C.

[31] T. Herlau, M. N. Schmidt, and M. Mørup. *Introduction to Machine Learning and Data Mining*. Lecture notes, Fall 2023, Technical University of Denmark (DTU). 2023. URL: https://www.compute.dtu.dk/english/education/courses/intro-ml.

[32] Historie Online. *Et af Danmarks ældste huse*. Accessed: 2025-05-22. n.d. URL: https://www.historie-online.dk/arkiv/danmarks-aeldste-hus.

[33] T. Hothorn et al. "Survival Ensembles". In: *Biostatistics* 7.3 (2006), pp. 355–373.

[34] H. Ishwaran et al. "Random Survival Forests". In: *The Annals of Applied Statistics* 2.3 (2008), pp. 841–860. DOI: 10.1214/08-AOAS169.

[35] ISO. *Buildings and constructed assets — Service life planning — Part 1: General principles*. 2011.

[36] K. Jager et al. "Confounding: What it is and how to deal with it". In: *Kidney International* 73.3 (Feb. 2008). Part of the ABC of Epidemiology series, pp. 256–260. DOI: 10.1038/sj.ki.5002657.

[37] A. K. Jensen et al. *I gamle dage byggede man udødelige bygninger eller hvad?* Accessed: 2024-11-25. 2024. URL: https://videnskab.dk/kultur-samfund/i-gamle-dage-byggede-man-udoedelige-bygninger-eller-hvad/.

[38] S. Ji, B. Lee, and M. Y. Yi. "Building life-span prediction for life cycle assessment and life cycle cost using machine learning: A big data approach". In: *Building and Environment* 205 (2021), p. 108267. DOI: 10.1016/j.buildenv.2021.108267.

[39] S. Jordan. "Roles of the reference service life (RSL) of buildings and the RSL of building components in the environmental impacts of buildings". In: *IOP Conference Series: Earth and Environmental Science*. Vol. 323. 1. IOP Publishing, 2019, p. 012146. DOI: 10.1088/1755-1315/323/1/012146.

[40] E. L. Kaplan and P. Meier. "Nonparametric Estimation from Incomplete Observations". In: *Journal of the American Statistical Association* 53.282 (1958), pp. 457–481. DOI: 10.2307/2281868. URL: https://doi.org/10.2307/2281868 (visited on 04/30/2025).

[41] J. L. Katzman et al. “DeepSurv: Personalized treatment recommender system using a Cox proportional hazards deep neural network”. In: *BMC Medical Research Methodology* 18.1 (2018), p. 24. DOI: 10.1186/s12874-018-0482-1.

[42] D. P. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations (ICLR)*. 2015. URL: https://arxiv.org/abs/1412.6980.

[43] Klimadatastyrelsen. *BBR Fildownload via HTTPS - Bygnings- og Boligregistret (BBR)*. https://datafordeler.dk/dataoversigt/bygnings-og-boligregistret-bbr/bbr-fildownload-https/. Accessed: May 22, 2025.

[44] Kræftens Bekæmpelse. *Asbest – tidligere anvendelse og sundhedsrisici*. Accessed: 2025-06-04. Kræftens Bekæmpelse. 2024. URL: https://www.cancer.dk/fakta-kraeft/aarsager/asbest/#Tidligere_anvendelsesomr%C3%A5der (visited on 06/04/2025).

[45] H. Kvamme. *pycox: Survival analysis with PyTorch*. Accessed: 2025-06-11. 2020. URL: https://github.com/havakv/pycox.

[46] H. Kvamme, Ø. Borgan, and I. Scheel. “Time-to-Event Prediction with Neural Networks and Cox Regression”. In: *Journal of Machine Learning Research* 20.129 (2019), pp. 1–30.

[47] M. LeBlanc and J. Crowley. “Survival trees by goodness of split”. In: *Journal of the American Statistical Association* 88.422 (1993), pp. 457–467. DOI: 10.2307/2290433.

[48] M. Lu and S. Ilgin Guler. “Comparison of Random Survival Forest with Accelerated Failure Time-Weibull Model for Bridge Deck Deterioration”. In: *Transportation Research Record* 2676.7 (2022), pp. 296–311. DOI: 10.1177/03611981221078281.

[49] M. Lunn. *Undergraduate lecture notes BS3b: Statistical Lifetime Models*. Tech. rep. Available at https://www.stats.ox.ac.uk/~mlunn/lecturenotes1.pdf. University of Oxford, Department of Statistics, 2007.

[50] R. Maniwa et al. “Boosting-Based Sequential Meta-Tree Ensemble Construction for Improved Decision Trees”. In: *arXiv preprint arXiv:2402.06386* (2024). DOI: 10.48550/arXiv.2402.06386. URL: https://arxiv.org/abs/2402.06386.

[51] C. Nadeau and Y. Bengio. "Inference for the generalization error". In: *Machine learning* 52.3 (2003), pp. 239–281.

[52] W. Nelson. "Theory and Applications of Hazard Plotting for Censored Failure Data". In: *Technometrics* 14.4 (1972), pp. 945–965. DOI: 10.1080/00401706.1972.10488991.

[53] C. Piech. *Probability for Computer Scientists*. https://chrispiech.github.io/probabilityForComputerScientists/en/intro/notation/. 2023.

[54] S. Pölsterl. "scikit-survival: A Library for Time-to-Event Analysis Built on Top of scikit-learn". In: *Journal of Machine Learning Research* 21.212 (2020), pp. 1–6. URL: http://jmlr.org/papers/v21/20-729.html.

[55] S. Pölsterl. "scikit-survival: A Library for Time-to-Event Analysis Built on Top of scikit-learn". In: *Journal of Machine Learning Research* 21 (2020), pp. 1–6.

[56] J. R. Quinlan. "Induction of Decision Trees". In: *Machine Learning* 1 (1986), pp. 81–106.

[57] S. Raschka. *mlxtend: A Python Library for Machine Learning Extensions*. https://rasbt.github.io/mlxtend/user_guide/evaluate/feature_importance_permutation/. Released under the New BSD License. Figures licensed under Creative Commons Attribution 4.0 International License. 2014.

[58] Regeringen and KL. *Gode Grunddata til alle - en kilde til vækst og effektivisering*. Archived from the original on October 8, 2012. Oct. 2012. URL: https://web.archive.org/web/20121008144427/http://fm.dk/nyheder/pressemeddelelser/2012/10/danmarks-digitale-raastof-saettes-fri/~/media/Files/Nyheder/Pressemeddelelser/2012/10/grunddata/WEB_Grunddatatilalle_v%C3%A6kstogeffektivisering_okt%202012.ashx (visited on 11/03/2012).

[59] G. Ridgeway. "Generalized Boosted Models: A Guide to the gbm Package". In: *Update* 1.1 (2007), pp. 1–12.

[60] S. M. Ross. *Introduction to Probability Models*. 11th. Academic Press, 2014.

[61] M. Rubin. "Inconsistent multiple testing corrections: The fallacy of using family-based error rates to make inferences about individual hypotheses". In: *Methods*

in Psychology 10 (Nov. 2024). ISSN: 2590-2601. DOI: 10.1016/j.metip.2024.100140. URL: http://dx.doi.org/10.1016/j.metip.2024.100140.

[62] K. Shedden and O. Mesner. *Statistics 504: Practice and Communication in Applied Statistics*. Course syllabus. Accessed: 2025-05-22. University of Michigan, Department of Statistics. 2025. URL: https://dept.stat.lsa.umich.edu/~kshedden/stats504/topics/syllabus/.

[63] H. Shima et al. "Estimation of Lifetime of Buildings and Prediction of Generation of Demolished Concrete by Proportional Hazard Model". In: *Journal of Environmental Engineering (Transactions of AIJ)* 68.573 (2003), pp. 87–94.

[64] Skatteministeriet. *Bekendtgørelse af lov om bygnings- og boligregistrering (BBR-loven)*. Lovbekendtgørelse. Retsinformation. Available at: https://www.retsinformation.dk/eli/lta/2019/797. 2019.

[65] M. Stone. "Cross-Validatory Choice and Assessment of Statistical Predictions". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 36.2 (1974), pp. 111–147. URL: http://www.jstor.org/stable/2984809 (visited on 05/07/2025).

[66] S. Szeghalmy and A. Fazekas. "A Comparative Study of the Use of Stratified Cross-Validation and Distribution-Balanced Stratified Cross-Validation in Imbalanced Learning". In: *Sensors* 23.4 (2023), p. 2333. DOI: 10.3390/s23042333. URL: https://www.mdpi.com/1424-8220/23/4/2333.

[67] T. Therneau and P. Grambsch. *Modeling Survival Data: Extending the Cox Model*. Springer, 2000.

[68] S. Varma and R. Simon. "Bias in error estimation when using cross-validation for model selection". In: *BMC Bioinformatics* 7.1 (2006), p. 91. DOI: 10.1186/1471-2105-7-91.

[69] H. Vestergaard and C. D. Haagerup. "The housing boom and bust in the Nordic and Baltic countries: Similarities and differences". In: *ENHR Conference 2012*. Paper presented at ENHR Conference 2012. Lillehammer, Norway, 2012. URL: http://www.congrex.no/enhr2012/.

[70] J. Wainer and G. Cawley. "Nested cross-validation when selecting classifiers is overzealous for most practical applications". In: *arXiv preprint arXiv:1809.09446* (2018). URL: https://doi.org/10.48550/arXiv.1809.09446.

[71] J. Wainer and G. C. Cawley. "Nested cross-validation when selecting classifiers is overzealous for most practical applications". In: *Expert Systems with Applications* 182 (2021). DOI: 10.1016/j.eswa.2021.115222.

[72] R. Williams. "Product-limit survival functions with correlated survival times". In: *Lifetime Data Analysis* 1.2 (1995), pp. 171–186. DOI: 10.1007/BF00985768.

[73] A. Zellner. "Erratum to "Generalizing the standard product rule of probability theory and Bayes's Theorem"". In: *Journal of Econometrics* 141.2 (2007). Original article: J. Econometrics 138 (1) (2007) 14–23, p. 1419. DOI: 10.1016/j.jeconom.2007.04.001.

[74] A. Zhang et al. *Generalization*. 2020. URL: https://d2l.ai/chapter_linear-regression/generalization.html (visited on 06/10/2025).

When Buildings Die

A Comparative Investigation of Machine Learning Approaches for Building Lifetime Modeling in Denmark