Forecasting Recipe Rank: Real-Time Prediction Framework

SN: 18006555[1]

MSc Data Science & Machine Learning

Project Supervisor: Dariush Hosseini

Submission date: 21st October 2024

[1] **Disclaimer:** This report is submitted as part requirement for the MSc Data Science & Machine Learning course at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except with permission from the author.

Abstract

Machine Learning applications are becoming increasingly prominent within retail businesses, for forecasting product sales or service adoption. The meal-kit delivery subdivision within the food industry presents the challenge of predicting the popularity of menu items in advance, for inventory management optimisation and food waste reduction. The UK hosts many subscription-based meal-kit startups, with one prominent player being 'Gousto', whose business model has potential to benefit from accurate forecasting of recipe uptake within their weekly menus. This study explores various statistical, supervised learning and learning-to-rank algorithms to predict the ranks of Gousto recipe uptakes in a given weekly menu, alongside the custom time-series experimental setup required to create effective long-term and short-term evaluation frameworks. The results indicate XGBoost as the superior long-term model with a 10.09% MAE 3 weeks prior to delivery. A piecewise LightGBM and Linear model attains a 2.49% MAE with a 7 day lag, dropping to 1.40% MAE 3 days in advance, constituting a short-term forecast which uses real-time customer selection data. This study concludes that learning-to-rank algorithms surprisingly underperform in this ranking task, including a state-of-the-art LambdaMART implementation via LightGBM Ranker, with suggested explanations and future optimisations required to better leverage the functionalities of ranking models. The source code for this study can be found at: .

Contents

Chapter 1

Introduction

1.1 Motivation

Gousto, a prominent UK-based meal-kit retailer, offers a dynamic weekly menu of diverse recipes for customers to choose from. Since its inception over a decade ago, the company has amassed a substantial dataset, containing both historical recipe data and recipe uptake patterns for each menu week. Within this business model, efficient inventory management is critical to ensuring that ingredients are available to fulfil customer demand, while minimising waste and optimising costs.

Given that customers have a minimum of three days to finalise their selections before delivery, there is a valuable opportunity for Gousto to benefit from more advanced forecasting techniques that allow for the prediction of final recipe selections well in advance of customer orders. Accurately forecasting recipe uptake weeks ahead of time would allow Gousto to improve ingredient sourcing and inventory management practices. Optimising this aspect of their operations could not only significantly reduce food waste but also lead to more cost-efficient procurement processes, enhancing the company's competitive positioning within the UK meal-kit industry.

This challenge motivates a machine learning-driven approach to predict recipe popularity across upcoming menu weeks. The focus of this study is on predicting recipe ranks as an alternative to direct uptake values, with the ultimate goal of integrating these predictions as meta-features within Gousto's broader uptake model. By improving the accuracy of recipe rank predictions, the overall model performance in forecasting uptake may be enhanced, thereby optimising inventory management and resource allocation.

1.2 Aims

The primary objective of this study is to explore and evaluate a range of machine learning models, including statistical methods, pointwise supervised learning algorithms, and learning-to-rank models, to predict recipe uptake ranks across different menu weeks. The dataset utilised consists of all recipes launched by Gousto, organised by menu week, and the models are tasked with predicting the relative popularity of each recipe within a given week. The evaluation framework incorporates both long-term and short-term forecasting approaches to assess performance at varying time lags.

For the long-term forecast, models are trained to predict future recipe uptake ranks using historical data and recipe features. Conversely, the short-term forecast focuses on real-time data that is added incrementally as features once the menu is released 18 days prior to the first delivery. This difference in prediction horizons necessitates distinct machine learning tasks and model evaluations, with the end goal of developing a piecewise model that optimises predictive performance throughout the entire span of a recipe's lifecycle.

The models, frameworks, and methodologies presented in this study are designed to provide a robust approach to time-series forecasting within Gousto's business operations, with broader applicability to related contexts within the food and retail industries.

1.3 Overview

Chapter 2 presents a comprehensive review of the current research landscape, examining the application of machine learning techniques within the food and retail sectors. Relevant papers on advanced machine learning methods and state-of-the-art ranking algorithms are also discussed, providing a theoretical foundation for the study.

Chapter 3 details the methodology employed in this study, outlining the key concepts, modelling approaches, and practical implications of applying machine learning to Gousto's business model. The dataset used for training and evaluation is described in detail, with a focus on time-series considerations and the challenges posed by the real-time nature of the short-term prediction task. Hypothesis tests are also initialised, to set up later uncertainty quantifications of final results.

Chapter 4 provides an in-depth discussion of the various statistical, pointwise, and ranking models evaluated in the study. The models' suitability for predicting recipe ranks is discussed in the context of the specific dataset characteristics, and key metrics such as Normalised Discounted Cumulative Gain (NDCG) are introduced for evaluation purposes.

Chapter 5 outlines the experimental design and implementation, including a detailed discussion of the engineering choices made to optimise the model training and evaluation process. The chapter also describes the hyperparameter tuning process and its role in improving model performance across both long-term and short-term forecasting tasks.

Chapter 6 presents the results of the experiments, comparing model performance across different time horizons and prediction tasks. Key insights and patterns are identified, with a focus on the strengths and limitations of each model type in the context of Gousto's business objectives. Hypothesis tests are conducted to statistically differentiate between model performances.

Chapter 7 summarises the study's key findings, offering a synthesis of the results and practical recommendations for model deployment. The chapter concludes by identifying the most effective models for predicting recipe ranks, highlighting the optimal approach for integrating these predictions into Gousto's broader data-driven decision-making processes.

Chapter 8 provides suggestions on study improvements and future considerations, discussing the acknowledgements of shortfalls and ways in which analysis could be extended. this covers the meta-feature backtest, ranking model optimisations and new feature generations.

Chapter 2

Related Work

This chapter reviews key research in the fields of demand forecasting and machine learning, with a particular focus on applications in the food supply chain, e-commerce, and ranking tasks. The following sections highlight advancements in predictive analytics and ranking models, which serve as foundational techniques for this study.

2.1 Machine Learning for Demand Forecasting

The application of machine learning to demand forecasting has been widely studied across industries, particularly in meal-kit services, supply chain management, and e-commerce. Smith et al. [2017] explored multinomial logistic regression and random forests to forecast customer behavior in meal-kit services, providing insights into how machine learning can improve inventory management in constrained platforms. They demonstrated that customer behavior models can predict recipe uptake with substantial accuracy.

Another example of machine learning in food demand forecasting is the work by Aci and Yergök [2023], who applied random forests to predict food demand in a university refectory. Their case study highlighted the strength of ensemble learning models in accounting for diverse features, which is key to handling complex datasets like those found in food services. Punia and Shankar [2022] took this further by using deep learning models, particularly LSTMs, to forecast demand in the food industry. They showed that deep learning excels in capturing long-term temporal dependencies in food demand.

Nagpal et al. [2024] introduced a novel approach for balancing convenience and nutrition in group meal recommendations through their BEACON system. Their multimodal approach to handling recipe features and customer preferences highlights how machine learning can be leveraged to forecast not just demand but also other relevant factors like nutritional content and customer satisfaction.

These studies demonstrate how machine learning techniques, ranging from traditional statistical models to advanced deep learning algorithms, can be effectively applied to demand forecasting in food supply chains. These methods are instrumental in managing inventory, reducing food waste, and optimising supply chains in environments where demand fluctuates significantly.

2.2 Advanced Forecasting Models

Advanced machine learning models, particularly ensemble methods and neural networks, have become the backbone of demand forecasting in dynamic industries. Panda and Mohanty [2023] highlighted the use of boosting techniques such as XGBoost in demand forecasting for food supply chains. They demonstrated that boosting methods, with their ability to capture intricate feature interactions, can significantly improve forecast accuracy.

Similarly, Adith et al. [2024] focused on explainable AI techniques and machine learning models like random forests for strategic analysis in food demand forecasting. Their work emphasised the importance of interpretability in business decision-making, showing that decision trees can provide the necessary transparency for operational strategies.

Additionally, Denis and Keerthana [2024] investigated ensemble learning techniques, including voting regressions, to improve demand forecasting. Their study reinforced the idea that combining different models often leads to more robust predictions, which can handle a wide range of scenarios in supply chains and food production.

These works underline the importance of advanced models such as gradient boosting, neural networks, and ensemble techniques, particularly in the context of complex and large-scale datasets. Their ability to adapt to fluctuating demand patterns is crucial for industries like retail and food services, where predicting future demand accurately is integral to operational efficiency.

2.3 Ranking Models in Machine Learning

Ranking models are vital in applications where the relative order of items is critical, such as search engines, recommendation systems, and e-commerce. Ranking methods, including pairwise and listwise approaches, have been widely studied and adopted. Pairwise ranking models, which compare two items at a time, have been used in applications such as recipe ranking in meal-kit services, where the goal is to order recipes by their predicted popularity. Listwise ranking models, on the other hand, evaluate entire lists, focusing on optimising the ranking of all items simultaneously [Burges, 2010].

Burges [2010] introduced *LambdaMART*, a combination of LambdaRank and gradient-boosted decision trees, as a more efficient way to optimise ranking tasks using pairwise comparisons. LambdaMART directly optimises metrics like Normalised Discounted Cumulative Gain (NDCG), which is crucial for applications where prioritising top-ranked items is key, such as in search engines and product recommendations.

In large-scale applications like *Bing* or *Yahoo!* search engines, LambdaMART has been shown to outperform traditional ranking models by efficiently handling large datasets while optimising for NDCG. Its ability to learn from pairwise item comparisons allows it to make accurate predictions about user preferences. In e-commerce, LambdaMART has been applied to recommend products based on past user interactions, improving user satisfaction by correctly ranking the most relevant items at the top of search results.

Though ranking models such as LambdaMART excel in specific ranking tasks, their utility can expand beyond search engines into other domains like food demand forecasting, where ranking recipes or products based on predicted popularity can drive significant operational improvements.

Chapter 3

Methodology

This chapter outlines the theoretical foundations and the methods applied in this study, providing a thorough description of the dataset and the approaches used for analysis.

3.1 Dataset

The dataset analysed in this study is a labelled tabular `JSON` dataset comprising 214 features and 27,330 rows, each corresponding to a recipe launched by Gousto between October 2014 and August 2024. The dataset's primary temporal feature is `menu_week`, representing the week of a recipe's launch, spanning from week 109 to week 619.

The number of recipes within each `menu_week` varies significantly, ranging from 10 recipes in week 109 to approximately 125 in week 618. The highest number of recipes occurs in menu week 579, which contains 155 recipes; this distribution is later analysed in figure 5.1.

The dataset includes 11 core features, such as `item_id`, which uniquely identifies a recipe, `average_rating`, which reflects customer feedback if the recipe has been previously launched, and `month`, capturing potential seasonality patterns.

Additionally, 202 features represent ingredient and attribute embeddings derived from the recipes' website descriptions and ingredients lists. These features are numerically transformed using Gousto's custom `Word2Vec` model, called `Recipe2Vec` [Tian et al., 2022], which ensures that similar recipes are positioned closely within a latent vector space. These are labelled as `att_emb_n` and `ing_emb_n`, representing recipe attributes and ingredients respectively, with n ranging from 0 to 99 for each.

Table 3.1 below displays a snapshot of the dataset, previewing the features and menu week structure within the data.

menu_week	item_id	average_rating	...	att_emb_0	...	ing_emb_0	...	recipe_uptake
109	312	4.5		0.12		0.45		0.67
109	323	3.8		-0.45		0.38		0.88
109	349	4.9		0.89		0.51		0.12
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
110	412	4.0		0.34		0.40		0.45
110	426	4.7		-0.22		0.47		0.90
110	434	3.9		0.78		0.39		1.09
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
619	512	4.6		0.73		0.60		1.64
619	543	4.3		0.40		0.55		0.54
619	559	4.2		0.11		0.62		0.05
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

Table 3.1: Snapshot of Dataset for Long-Term Model. Table contains placeholder values for all features but `menu_week`. There are 11 base features, 202 embedding features and the original target variable. only the first 3 recipes per menu week are displayed.

3.1.1 Rank Percentiles

The initial target variable, `recipe_uptake`, represents the popularity of a recipe within its respective `menu_week` as a continuous positive value. While `recipe_uptake` is the primary target for Gousto's prediction models, this study introduces an alternative approach by incorporating the predicted rank of a recipe as a meta-feature. Thus, the original `recipe_uptake` is replaced by an ordinal rank (`recipe_rank`) within each week, where 0 and $n-1$ represent the highest and lowest uptake, respectively.

Given the varying number of recipes per week, the ranks are normalised to calculate the `recipe_rank_percentile`, scaling the values between 0 and 1, representing the lowest and highest uptake. This normalised rank percentile becomes the primary target variable for most models discussed in this study.

3.2 Exploratory Data Analysis

This section details the exploratory data analysis performed on the dataset's features and target variables, with a focus on both the long-term and short-term models.

3.2.1 Long-term Dataset

For the long-term model, a correlation analysis was carried out between all core features, two embedding meta-features, and `recipe_rank_percentile`. The results of this analysis, ordered by correlation strength, are presented in Figure 3.1.

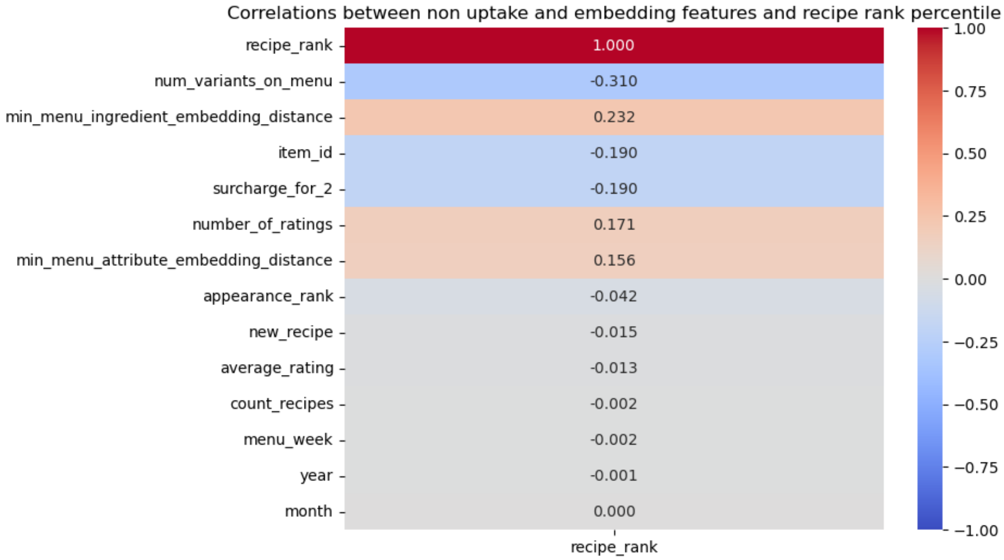

Figure 3.1: Heatmap of correlations between core features, two embedding meta-features, and `recipe_rank_percentile`

As shown in Figure 3.1, none of the core features exhibit strong linear correlations with the target variable, `recipe_rank_percentile`. This lack of clear linear relationships suggests the necessity for more sophisticated machine learning models capable of capturing non-linear dependencies in the data. It is important to note that embedding features were excluded from the plot, as their direct interpretation is limited due to their transformation by the `Recipe2Vec` encoder. Regardless, although not visualised, the correlation values for these features indicated weak or spurious relationships, with values around ±0.2.

3.2.2 Short-term Dataset

For the short-term model, a correlation analysis was performed between all `uptake_at_lead_day_n` features and `recipe_rank_percentile`. The results, displayed in Figure 3.2, are shown to eight decimal places to capture the high precision needed to distinguish between the correlation values.

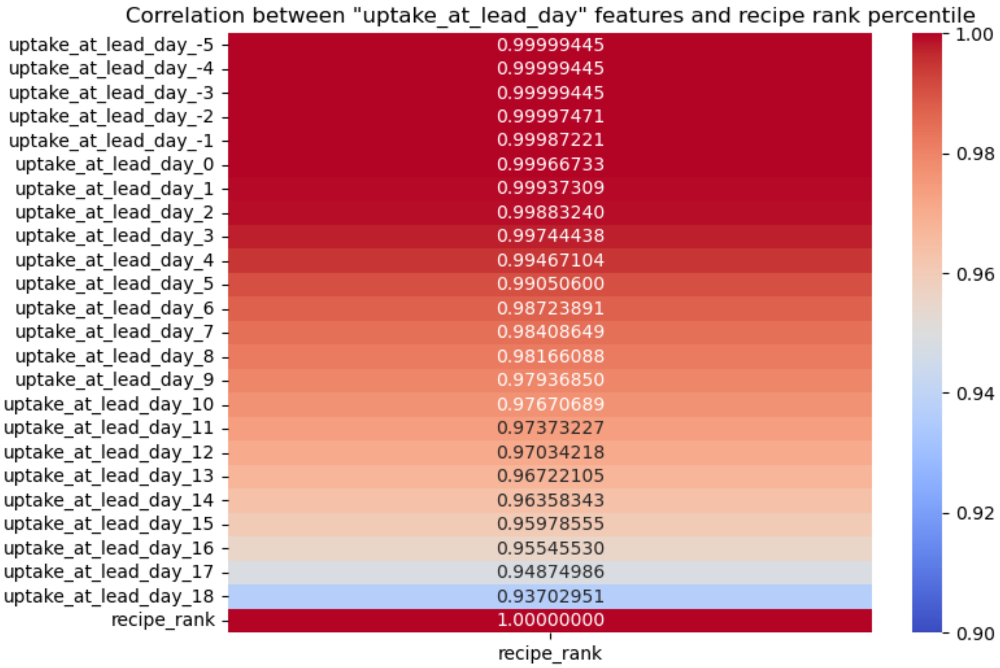

Figure 3.2: Heatmap of correlations between all `uptake_at_lead_day` features and `recipe_rank_percentile`

Figure 3.2 reveals strong linear correlations between the real-time uptake features, with the correlation values steadily approaching 1.0 as the `lead_day` decreases. Notably, even on `lead_day_18`, the correlation coefficient remains high at 0.937, marking this as a crucial feature despite the 18-day gap before delivery. This finding supports the use of Linear Regression for the short-term model and validates the assumption that this feature converges smoothly towards the target.

3.3 Evaluation Framework

This section outlines the importance of establishing an effective evaluation framework tailored to Gousto's requirements and timelines.

3.3.1 Evaluation Metric

The models are evaluated using the mean absolute error (MAE) [Willmott and Matsuura, 2005], which measures the average magnitude of the differences between predictions and their corresponding true values across all recipes in the evaluation subset. The formula for MAE is given below, where $\hat{y}_i$ represents the predicted value and y_i is the true value:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |\hat{y}_i - y_i|$$

MAE has been chosen for several reasons. Firstly, when these rank predictions are used as features within Gousto's uptake model, it is logical to select an error metric similar to Gousto's own, which is a weighted mean absolute percentile error (WMAPE). The absolute error nature of MAE aligns with this, while the use of rank percentiles as targets ensures that percentile errors are naturally accounted for.

Secondly, evaluating with MAE, rather than metrics like mean squared error (MSE), enhances the interpretability of model performance. For instance, an MAE of 0.10 suggests that a recipe's predicted rank is off by approximately five positions on average in a menu week of 50 recipes. This level of interpretability is essential for understanding the practical implications of model errors within a business context.

3.3.2 Multi-Week Forecasting Framework

The performance of the long-term model is evaluated using time series principles, as each new menu week follows chronologically from the previous one. This necessitates careful preprocessing to prevent data leakage, ensuring that no future menu weeks are used to predict recipes in the past. To achieve this, the training data is always at least one menu week behind the test data. Additionally, menu weeks are processed in batches so that recipes from the same week are not split between training and testing datasets. In testing, only one menu week is used, mirroring real-world production scenarios where a specific menu is predicted at a time.

A key characteristic of the evaluation framework is the multi-week forecasting approach, where model performance is assessed several weeks before the target week. Gousto finalises the recipes for a menu up to eight weeks before the delivery date, roughly five weeks before the menu goes live for customer selection. The long-term model supports early forecasting, allowing predictions between 18 days and 8 weeks before the menu week launch. Therefore, the model's forecasting ability is evaluated by limiting the training data to k weeks before the target week, where $k \in \{1, 2, \cdots, 8\}$, with $k = 0$ representing the test week. This enables plotting model performance (MAE) over time, with the expectation that MAE will decrease as the training data gets closer to the target week. Consequently, the goal shifts from finding a single best-performing model to potentially combining models, where one excels at predicting 8 weeks out, and another performing better closer to the target.

Robustness is ensured by averaging model evaluation across 52 iterations, where the test week is cycled through the most recent 52 menu weeks. For each test week, the model is trained eight times, once for each training lag described. This setup mitigates the impact of seasonality on evaluation and allows assessment of the model's ability to predict k weeks in advance for any given test week. Details of the Multi-Week pipeline are provided in section 5.3.

3.3.3 Uncertainty Quantification

Evaluating model performance across 52 iterations not only enhances robustness by computing the mean MAE but also facilitates uncertainty quantification through the standard error. A 95% confidence interval is calculated from these iterations, capturing two standard errors from the mean, enabling a comparison of uncertainty between models [Efron and Tibshirani, 1994]. This provides a new dimension for model evaluation, as both performance (MAE) and uncertainty are considered. The 95% confidence intervals are visualised with error bars around each mean, where the width of the error bars represents the confidence interval.

Hypothesis tests are conducted within this study to further quantify model comparisons via paired t-tests, used alongside the aforementioned confidence intervals. These will provide more concrete justifications between selecting the optimal models from the obtained results, leveraging the 52 iterations of each experiment. These tests are defined and explored further in section 3.6, later in this chapter.

A notable alternative for estimating the distribution of a statistic is the bootstrap method, which involves resampling the dataset to construct confidence intervals. However, this approach may not be suitable for ranking contexts where distinct rankings are crucial. The introduction of duplicates can result in ties, complicating the evaluation of metrics like NDCG (discussed in section 4.4), which relies on a unique ordering of items. In ranking systems, the inability to differentiate between similar items can misrepresent their true importance and mislead evaluations. Therefore, a traditional method of constructing 95% confidence intervals is preferred, as it preserves the integrity of the ranking system and provides a more intuitive assessment of model performance.

3.4 Hyperparameter Tuning

Hyperparameter tuning is a critical step in optimising model performance by identifying the most suitable parameters for each model. In this study, the tuning process was conducted using Optuna, a cutting-edge hyperparameter optimisation framework that employs a Tree-structured Parzen Estimator (TPE) to efficiently explore the hyperparameter space [Akiba et al., 2019]. Tuning was performed on a validation set generated via time series 3-fold cross-validation (CV), which preserves temporal order and prevents data leakage, further detailed in section 5.4.1.

Optuna was chosen for its efficiency in navigating large and complex hyperparameter spaces through TPE, a Bayesian optimisation method. TPE constructs probabilistic models to identify promising hyperparameter values, focusing the search on areas of the space that are more likely to yield optimal results based on prior trials [Bergstra et al., 2011]. This approach significantly improves the tuning process compared to traditional methods such as grid search or random search.

Hyperparameter tuning for each model was carried out over multiple iterations with the objective of minimising the validation MAE. Optuna was run for a predefined number of trials, ensuring that the selected hyperparameters optimise the model's predictive performance.

Additionally, tuning was applied to the `window_size` hyperparameter for each model, which defines the number of menu weeks included in the training set [Bergmeir et al., 2018]. This parameter allows models to improve performance by training on only the most recent W menu weeks, where W is the window size, rather than on the entire dataset. This tuning process is further detailed in section 5.4.2.

3.5 Real-Time Predictions

The short-term model focuses on predicting recipe rank percentiles using real-time data as a feature for the 18 days leading up to the final delivery date. These predictions build on the base predictions from the long-term model, made between 0 to 3 weeks before the test week, with the aim of significantly improving model performance with the inclusion of real-time data, given the appropriate handling of short-term features.

3.5.1 Real-Time Data

An additional 24 features are introduced alongside the existing features from Section 3.1, capturing real-time data from customers' live recipe selections. These features are denoted as `uptake_at_lead_day_n`, where $n \in \{18, 17, \ldots, -4, -5\}$, with n representing the number of days before the first possible delivery date (at $n = 0$). These features are similarly transformed into rank percentiles to align with the target variable. It is important to note that real-time data is only available for recipes from `menu_week` 488 onwards, significantly limiting the training data.

Gousto releases their menu 18 days before the first delivery, allowing customers to pre-select and modify their recipe choices up to 3 days before their chosen delivery date. These features are crucial for prediction as their values converge towards the target over time. For example, `uptake_at_lead_day_3` is expected to closely align with the target as customers begin locking in their selections, especially after receiving reminder emails 1-2 days prior. Any remaining discrepancies are due to customers choosing later delivery dates within a given menu week. The last three days of `lead_day_k` for $k \in \{-4, -5, -6\}$ will have identical values to `uptake_at_lead_day_k+1` since choices are locked in 3 days before the final possible delivery on `lead_day_6`.

Lead days are also labelled as `lead_day_m`, where $m \in \{17, 16, \ldots, -5, -6\}$. Customers can select their delivery day from `lead_day_0` to `lead_day_-6`. However, there is a 1-day lag between customers' recipe selections and the availability of this data for predictions. Thus, while `uptake_at_lead_day_n` records uptake for lead day n, it is only usable for predictions on lead day $n-1 = m$. As a result, predictions made on `lead_day_n-1` use data from `uptake_at_lead_day_n`.

To further enhance the model's ability to capture trends in recipe selection, 23 additional short-term features are generated through feature engineering [Zheng and Casari, 2018]. These features represent the changes in recipe uptake between consecutive lead days and are labelled as `diff_uptake_at_lead_day_n`. Each `diff_uptake_at_lead_day_n` reflects the difference in uptake between lead days $n + 1$ and n. As with the uptake features, these difference features are only available for prediction on lead day $n - 1$. A snapshot of the short-term dataset can be previewed in table 3.2 below:

menu_week	item_id	...	uptake_18	diff_upt_17	uptake_17	...	diff_upt_-5	uptake_-5	recipe_uptake
488	312	...	0.55	0.10	0.45	...	0.15	0.60	0.60
488	323	...	0.50	0.05	0.45	...	0.10	0.50	0.50
488	349	...	0.60	0.15	0.45	...	0.20	0.55	0.55
⋮	⋮	...	⋮	⋮	⋮	...	⋮	⋮	⋮
489	412	...	0.48	0.05	0.43	...	0.05	0.45	0.45
489	426	...	0.49	0.01	0.48	...	0.08	0.47	0.47
489	434	...	0.40	-0.02	0.42	...	0.06	0.41	0.41
⋮	⋮	...	⋮	⋮	⋮	...	⋮	⋮	⋮
619	512	...	0.60	0.05	0.55	...	0.15	0.60	0.60
619	543	...	0.55	0.03	0.52	...	0.10	0.55	0.55
619	559	...	0.62	0.02	0.60	...	0.20	0.62	0.62
⋮	⋮	...	⋮	⋮	⋮	...	⋮	⋮	⋮

Table 3.2: Snapshot of Dataset for Short-Term Model. Column headings have been shortened, with highlights on the `menu_week` and `item_id` indices, and the real-time data and differences.

By incorporating both uptake and difference features, the model captures not only the current level of recipe uptake but also the direction and magnitude of changes in customer behaviour over time.

3.5.2 Multi-Day Forecasting Framework

The short-term model is evaluated across all `lead_days` to predict the final target within `lead_day_k` for $k \in \{-4, -5, -6\}$. Each additional day introduces new columns of data, including `uptake_at_lead_day_n` and `diff_uptake_at_lead_day_n`, which are used on `lead_day_n-1` but represent customer data from `lead_day_n`. The forecast horizon is extended to 21 days (3 weeks) before the first delivery date (`lead_day_0`), allowing better visualisation of the model's improvement as short-term real-time uptake features are gradually incorporated alongside long-term features.

Similar to the long-term model, predictions are generated for each `lead_day_n` where $n \in \{21, 20, \cdots, -5, -6\}$. The model is evaluated over 52 iterations to account for potential seasonality and enhance robustness. Standard error bars are used to compute a 95% confidence interval for each lead day. Note that predictions on `lead_day_n` for $n \in \{21, 20, 19, 18\}$ rely solely on long-term features without the inclusion of real-time data. Predictions on `lead_day_n` for $n \in \{-4, -5, -6\}$ remain the same due to the 3-day period during which customer choices are locked in, resulting in identical real-time data (`uptake_at_lead_day_n+1`).

Regarding long-term features, at `lead_day_17`, recipes from three menu weeks prior to the target week are used, with additional rows from more menu weeks included as the lead day progresses. This follows a simple conversion from weeks to days, whereby every 7 days a new menu week is added to the training data. Therefore as the lead day advances, both rows (menu weeks) and columns (features) are added to the training dataset. Further details on the setup of the Multi-Day pipeline are provided in Section 5.5.2.

3.6 Hypothesis Tests

One major aim within this study includes deciding which model provides the best MAE performance in the long-term and short-term models. Whilst a simple comparison of MAEs between two or more models may provide a strong indicator, alongside the aggregation over 52 iterations per model for each lag size for robustness, it is still crucial to quantify the uncertainty of any conclusions drawn from generated results.

A hypothesis test is a statistical method used to make inferences about a population based on sample data. It involves formulating two competing hypotheses: the null hypothesis (H_0), which represents the status quo or a statement of no effect, and the alternative hypothesis (H_1), which reflects the study's aim or the effect the study is attempting to support. The test evaluates the evidence provided by the data to either reject the null hypothesis in favor of the alternative or fail to reject the null hypothesis.

3.6.1 Long-Term Hypothesis Tests

In the context of this study, the initial presumption is that ranking models should outperform pointwise models when evaluating MAE. This assumption is rooted in the design of ranking models, which are tailored for tasks where the order of predictions is crucial compared to pointwise models that typically assess each observation independently. This could suggest a one-tailed approach, however in order to account for all possibilities, a two-tailed hypothesis test is selected.

For the long-term models, there will be 2 hypothesis tests conducted, with three iterations of each test for each selected lag size of 8 weeks, 3 weeks, and 1 week. This mirrors the structure of the results chapter, since these are important milestones for Gousto's menu week cycle. The hypotheses for these tests are as follows:

Comparison of the Singular Pointwise Model vs. the Best Ranking Model

This first test will compare the best performing ranking and pointwise models to understand if there is a significant disparity between both. The baseline statistical models are excluded since these are naive models and are not expected to significantly outperform the best pointwise and ranking models. The sign of the t-statistic will be used to determine which model performs better in this test.

- Null Hypothesis (H_0): The best ranking model performs equally as well as the best pointwise model.

$$H_0 : \text{MAE}_{\text{Best Ranking Model}} = \text{MAE}_{\text{Best Pointwise Model}}$$

- Alternative Hypothesis (H_1): There is a significant disparity between the performance of the best ranking model and the best pointwise model.

$$H_1 : \text{MAE}_{\text{Best Ranking Model}} \neq \text{MAE}_{\text{Best Pointwise Model}}$$

Comparison of the LightGBM Ranker vs. the Candidate Model

The next test involves a comparison between LightGBM Ranker (LambdaMART) and a chosen candidate model. The Candidate Model will be the best performing model from the previous test unless it is the LightGBM Ranker itself. LambdaMART is considered a state-of-the-art model for ranking tasks due to its ability to effectively leverage gradient boosting techniques. It is expected to deliver superior performance compared to other models in this study, making it a critical point of comparison. The sign of the t-statistic will indicate which model performs better.

- Null Hypothesis (H_0): The LightGBM Ranker performs equally as well as the Candidate Model.
$$H_0 : \text{MAE}_{\text{Candidate Model}} = \text{MAE}_{\text{LightGBM Ranker}}$$
- Alternative Hypothesis (H_1): There is a significant disparity between the performance of the LightGBM Ranker and the Candidate Model.
$$H_1 : \text{MAE}_{\text{Candidate Model}} \neq \text{MAE}_{\text{LightGBM Ranker}}$$

3.6.2 Short-Term Hypothesis Tests

In addition to the long-term analysis, short-term hypothesis tests will be conducted to compare the top two performing models at various `lead_days` ranging from `lead_day_17` to `lead_day_-6`. Another test will be conducted between the best performing model (or a crafted piecewise model) decided by the previous test, against using the real-time data from customer selections as a prediction in itself, to quantify the benefit of using machine learning within the short-term forecast.

For each day before the first delivery date, the test will provide uncertainty quantifications to understand the best model for each, facilitating the potential deployment of a piecewise model for the entire short-term time span.

Model A and B are chosen as the best two models within the short-term model analysis. Here, the focus will be on conducting a single hypothesis test between Model A and Model B for each time stamp within the above range, across 52 menu week iterations. This will result in 24 tests in total.

Comparison of Model A vs. Model B

In this test, an assessment will be made as to whether there is a significant disparity between the performances of Model A and Model B. The sign of the t-statistic will help us determine which model is superior.

- Null Hypothesis (H_0): Model A performs equally as well as Model B.
$$H_0 : \text{MAE}_{\text{Model A}} = \text{MAE}_{\text{Model B}}$$
- Alternative Hypothesis (H_1): There is a significant disparity between the performance of Model A and Model B.
$$H_1 : \text{MAE}_{\text{Model A}} \neq \text{MAE}_{\text{Model B}}$$

Comparison of Best Model vs. Real-Time Feature Model

In this test, an assessment will be made as to whether there is a significant disparity between the performances of the Best Model decided by the previous test and the model which uses the real-time uptake data as predictions. The sign of the t-statistic will help us determine which approach is superior.

- Null Hypothesis (H_0): The Best Short-Term Model performs equally as well as the Real-Time Feature Model.

$$H_0 : \text{MAE}_{\text{Best Model}} = \text{MAE}_{\text{Real-Time Model}}$$

- Alternative Hypothesis (H_1): There is a significant disparity between the performance of the best short-term model and the real-time feature model.

$$H_1 : \text{MAE}_{\text{Best Model}} \neq \text{MAE}_{\text{Real-Time Model}}$$

3.6.3 Hypothesis Test Summary

By employing these hypothesis tests, the study aims to provide a rigorous statistical framework for assessing the performance differences between the evaluated models over both long-term and short-term forecasting horizons. The results will help clarify whether the observed differences in MAE are statistically significant, thereby offering insights into the effectiveness of ranking versus pointwise models in various forecasting contexts. The experimental setup for all hypothesis tests is described in section 5.6.

Chapter 4

Models

4.1 Statistical Models

This section introduces three baseline statistical models used for predicting recipe rank percentiles across different menu weeks. These models serve as benchmarks for evaluating more advanced predictive methods, offering approaches that range from simple averages to methods based on recent or cumulative recipe occurrences.

While highly interpretable, these models treat each recipe independently, ignoring the competitive dynamics within a menu week. Despite this limitation, their simplicity and transparency make them essential for establishing baseline performance.

4.1.1 Total Average Model

The *Total Average Model* serves as a simple yet effective baseline in this study, predicting the target value as the average of all historical observations, namely the rank predictions of all recipes in prior menu weeks. This model operates under the assumption that there are no significant temporal trends or seasonal variations, making it a straightforward benchmark [Makridakis et al., 1998]. The below formula demonstrates the calculation required for prediction, where D is the training dataset, y_i is each target within D, indexed by the ith recipe in D.

$$\hat{y} = \frac{1}{|D|} \sum_{i \in D} y_i$$

Despite its simplicity, the Total Average Model plays a crucial role in the evaluation process, as it sets a minimum standard for model performance. If a more sophisticated model fails to outperform this baseline, it signals potential issues in that model's design or training. As such, this benchmark helps to ensure the reliability and robustness of the more advanced predictive models under investigation.

4.1.2 Last Occurrence Model

The *Last Occurrence Model* predicts the recipe's rank percentile based on the most recent observation of the recipe, identified by its `item_id`. In literature, this is commonly known as the naive forecast. This model assumes that the latest performance of the recipe carries the greatest predictive weight for future outcomes, making it especially useful in datasets where recipe ranks exhibit low variability or where recent trends are highly indicative of future behaviour [Petropoulos and Kourentzes, 2015]. The below piecewise formula shows the prediction calculation, where L is the index of the recipe within the latest menu week where the recipe was listed.

$$\hat{y} = \begin{cases} y_L & \text{if the recipe appeared before } (\exists L), \\ \frac{1}{|D|} \sum_{i \in D} y_i & \text{otherwise.} \end{cases}$$

If the recipe has not appeared before in the data, the model defaults to the average rank percentile across the training dataset, effectively aligning with the Total Average Model. This approach ensures robustness by handling both frequently occurring recipes and those that may have sparse historical data, providing a straightforward yet reliable benchmark for more complex forecasting methods.

4.1.3 Average Occurrence Model

The *Average Occurrence Model* predicts the rank percentile of a recipe by averaging all previous occurrences of that recipe, identified by its `item_id`, across all available menu weeks [Hyndman and Athanasopoulos, 2018]. With $P \subseteq D$ defined as the subset of prior occurrences by the same recipe, the predicted rank percentile $\hat{y}$ for a given recipe is defined as:

$$\hat{y} = \begin{cases} \frac{1}{|P|} \sum_{j \in P} y_j & \text{if the recipe has appeared before } (|P| > 0), \\ \frac{1}{|D|} \sum_{i \in D} y_i & \text{otherwise.} \end{cases}$$

By averaging the rank percentiles across all instances of a given recipe, this model provides a more comprehensive view of the recipe's historical performance. It smooths out fluctuations that may occur in individual menu weeks, offering a stable and reliable forecast for future rankings. This method is particularly effective for recipes that exhibit variability in rank performance over time. If a recipe has never appeared before, the model defaults to the average rank percentile of all recipes in the training dataset, ensuring robustness and consistency with other benchmark models.

4.2 Supervised Learning Models

4.2.1 Linear Regression

Linear Regression is one of the simplest and most interpretable models in machine learning. It assumes a linear relationship between the input features and the target variable [James et al., 2013]. The model predicts the target value $\hat{y}$ as a weighted sum of the input features:

$$\hat{y} = \mathbf{w}^\top \mathbf{x} + b$$

where $\mathbf{w}$ represents the learned weight vector, $\mathbf{x}$ is the input feature vector, and b is the bias term. The objective of linear regression is to learn the weights and bias by minimising the difference between the predicted and actual target values. This study achieves this through optimisation of the Mean Absolute Error (MAE) loss function:

$$\text{MAE} = \frac{1}{|D|} \sum_{i \in D} |y_i - \hat{y}_i|$$

where y_i is the true value of the target variable for the i-th instance, and $\hat{y}_i$ is the corresponding predicted value. Once again, D represents the training set. The MAE is chosen in training ahead of the more common use of the Mean Squared Error (MSE) with Linear Regression, due to study's evaluation metric also being the MAE.

Linear Regression is often used as a baseline for regression tasks due to its simplicity and ease of interpretation. The learned weights $\mathbf{w}$ provide insights into the importance and influence of each feature on the target variable. Additionally, its closed-form solution and efficient computation make it a popular choice for many real-world applications.

In this study, since the model has significantly more rows (recipes) than features (27730×214 in the long-term model), it forms an over-determined system. This configuration generally improves the model's stability and ensures that the parameter estimates are well-constrained. The model benefits from having sufficient data to fit the parameters without risking overfitting or instability, which can occur in under-determined systems where there are more features than rows.

However, despite its interpretability, Linear Regression has limitations. The primary drawback is its assumption of a linear relationship between the input features and the target variable. This makes the model unsuitable for capturing complex, non-linear patterns in the data. In such cases, more advanced models that can handle non-linearity are necessary to improve predictive performance. The exploratory data analysis in section 3.2 suggested that linear regression may not provide a strong model performance due to the lack of high feature correlations with the target within the long-term model, however may prove impactful within the short-term model. Additionally, Linear Regression is sensitive to outliers, which can disproportionately affect the learned weights and bias.

Overall, Linear Regression provides a useful starting point for predictive modeling, but its limitations in handling non-linear relationships and susceptibility to outliers highlight the need for more sophisticated methods in certain contexts.

4.2.2 LightGBM

LightGBM (Light Gradient Boosting Machine) is a highly efficient gradient boosting framework developed for fast training, low memory usage, and scalability, especially suited for large datasets and high-dimensional feature spaces [Ke et al., 2017, Chen and Guestrin, 2016]. It builds an ensemble of decision trees in a stage-wise manner, where each successive tree attempts to minimise the residual errors from the previous iterations. This technique forms the basis of gradient boosting, which LightGBM optimises further with several unique approaches.

The objective function of LightGBM, tailored for this study, can be expressed as:

$$\mathcal{L} = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{j=1}^{K} \Omega(f_j)$$

where $l(y_i, \hat{y}_i)$ represents the Mean Absolute Error (MAE) loss function defined as:

$$l(y_i, \hat{y}_i) = |y_i - \hat{y}_i|$$

Here, $\hat{y}_i$ are the predicted values, and y_i are the true values. The second term, $\Omega(f_j)$, is the regularisation term for tree j that helps prevent overfitting and is typically defined as:

$$\Omega(f) = \lambda \sum_{j=1}^{K} ||w_j||_1 + \frac{\gamma}{2} \sum_{j=1}^{K} ||w_j||_2^2$$

where w_j are the weights of the leaves in tree j, and λ and γ are hyperparameters that control the strength of the regularisation [Chen and Guestrin, 2016].

One of the most notable innovations in LightGBM is its leaf-wise growth strategy. In contrast to the level-wise approach used by traditional gradient boosting methods, where trees expand level by level, LightGBM grows trees by splitting the leaf that maximises the loss reduction, defined as:

$$\text{Gain}(t) = \frac{1}{2} \left(\frac{(G_L + G_R)^2}{H_L + H_R + \lambda} - \frac{G_L^2}{H_L + \lambda} - \frac{G_R^2}{H_R + \lambda} \right)$$

where G_L and G_R are the sums of gradients for the left and right child nodes, and H_L and H_R are the sums of Hessians for the left and right child nodes. This approach allows the algorithm to focus on reducing the largest errors first, resulting in deeper and more effective trees. This leaf-wise strategy typically leads to better accuracy, although it may produce unbalanced trees [Ke et al., 2017].

Additionally, LightGBM implements histogram-based learning, where continuous features are discretised into a set of bins, significantly reducing the computational cost of finding the optimal split points. The discretisation can be formulated as follows:

$$\text{Bin}(x) = \left\lfloor \frac{x - \min}{\text{bin\_size}} \right\rfloor$$

This binning process accelerates both training speed and memory efficiency by grouping similar values together, which is particularly important when working with large datasets or features with high cardinality. Moreover, the model can natively handle categorical features without requiring one-hot encoding, streamlining the preprocessing pipeline and saving memory and computation costs [Ke et al., 2017].

Regularisation is an integral part of LightGBM to prevent overfitting, especially when working with complex data. The model employs both L_1 (Lasso) and L_2 (Ridge) regularisation, which penalise the complexity of the model and help control the trade-off between bias and variance. Techniques such as early stopping, which halts training once performance on a validation set stops improving, and the ability to tune tree depth further enhance its robustness against overfitting [Chen and Guestrin, 2016].

In this study, LightGBM is used to predict recipe rank percentiles across menu weeks, optimising the Mean Absolute Error (MAE) as the loss function. Its ability to handle large datasets with numerous features, such as historical recipe ranks, customer preferences, and recipe characteristics, makes it ideal for modelling non-linear relationships and capturing intricate feature interactions in the data.

Overall, LightGBM's combination of efficiency, flexibility, and scalability, along with its sophisticated methods for improving predictive performance and avoiding overfitting, makes it a highly suitable candidate for the recipe ranking problem tackled in this study.

4.2.3 XGBoost

XGBoost (Extreme Gradient Boosting) is a powerful and flexible gradient boosting framework that has gained popularity for its performance in various machine learning tasks, including regression and classification [Chen and Guestrin, 2016]. It employs an ensemble learning technique that constructs multiple decision trees, where each tree attempts to correct the errors made by its predecessors. In this study, the Mean Absolute Error (MAE) loss function is utilised during training, providing an interpretable measure of error robust to outliers.

The objective function in XGBoost, specifically when using the MAE loss, can be expressed as follows:

$$\mathcal{L} = \sum_{i=1}^{n} |y_i - \hat{y}_i| + \sum_{j=1}^{K} \Omega(f_j)$$

where $|y_i - \hat{y}_i|$ is the MAE measuring the difference between the predicted values $\hat{y}_i$ and the true values y_i, $\Omega(f_j)$ represents the regularisation term, and K denotes the total number of trees. This formulation helps ensure robust predictions even in the presence of incomplete records [Friedman, 2001].

Moreover, XGBoost incorporates regularisation techniques, including L_1 (Lasso) and L_2 (Ridge) penalties, defined as:

$$\Omega(f) = \lambda \sum_{j=1}^{K} ||w_j||_1 + \frac{\gamma}{2} \sum_{j=1}^{K} ||w_j||_2^2$$

where w_j are the weights of the leaves in tree j, and λ and γ are hyperparameters controlling the strength of the regularisation. These regularisation methods help control model complexity and prevent overfitting, which is critical in regression tasks where the model must generalise across various menu weeks. By penalising overly complex models, XGBoost ensures that it captures the most relevant features while discarding noise, ultimately enhancing its predictive reliability across diverse scenarios.

Another key feature of XGBoost is its implementation of tree pruning and shrinkage. During the tree construction process, XGBoost employs a technique known as "max depth" and "minimum child weight", which stops the growth of a tree once it reaches a certain depth or when additional splits do not improve the predictive power significantly. This process, combined with shrinkage (scaling the contributions of new trees by a learning rate η), is expressed mathematically as:

$$\hat{y}_{new} = \hat{y}_{old} + \eta f(x)$$

where $\hat{y}_{new}$ is the updated prediction, $\hat{y}_{old}$ is the previous prediction, and $f(x)$ is the prediction from the new tree. This ensures that the model remains robust without being overly sensitive to training data [Chen and Guestrin, 2016].

Overall, XGBoost is well-suited for this study to predict recipe rankings across menu weeks due to its robust handling of large, sparse datasets, capacity to capture complex feature interactions, and effective management of model complexity. Its efficiency and flexibility make it an ideal tool for delivering accurate and scalable predictions in dynamic environments. XGBoost remains a strong candidate in this study for its proven performance and adaptability in diverse applications.

4.2.4 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and aggregates their predictions to enhance the robustness of recipe rank percentile predictions. The algorithm operates on the principle of 'bagging' (bootstrap aggregating), where multiple subsets of the training data are randomly sampled with replacement to train individual trees. Each tree is built using a random selection of features at each split, which introduces diversity among the trees and mitigates the risk of overfitting [Breiman, 2001].

The training process begins with bootstrap sampling, which generates diverse subsets from the original dataset, ensuring that each tree learns from a slightly different perspective. For each subset, a decision tree is constructed by recursively splitting nodes based on the selected features. The quality of these splits is typically assessed using criteria such as the Gini impurity for classification tasks or the Friedman Mean Squared Error (MSE) for regression tasks. The Friedman MSE, specifically, is expressed as:

$$\text{MSE} = \frac{1}{|D|} \sum_{i \in D} (y_i - \hat{y}_i)^2$$

where y_i represents the true values, $\hat{y}_i$ indicates the predicted values, and D denotes the training set. This criterion adjusts for node variance by prioritising splits that effectively reduce error while considering the overall structure of the tree [Friedman, 2001]. Once all trees are constructed, their predictions are aggregated by averaging for regression tasks, which enhances the overall prediction robustness.

Random Forest's capability to manage large datasets and capture complex, non-linear relationships makes it particularly well-suited for recipe ranking. By averaging predictions across multiple trees, the model reduces overfitting and increases stability, especially in cases where the dataset may have sparse or missing values. The use of surrogate splits allows the algorithm to handle these challenges effectively, while its inherent feature selection capabilities help to manage datasets with numerous features [Breiman, 2001].

Despite these advantages, a potential drawback of Random Forest in this context is its lack of inherent boosting, which can lead to less precise predictions, particularly when addressing complex residual errors that boosting methods effectively correct. Additionally, the training metric of Friedman MSE may conflict with the evaluation metric of Mean Absolute Error (MAE), as the former emphasises variance reduction while the latter focuses on the accuracy of individual predictions. This discrepancy might result in a model that excels during training but fails to generalise well to the evaluation metric.

Nonetheless, Random Forest's ability to capture non-linear relationships and its robustness against overfitting make it a compelling model for recipe ranking. Its proficiency with diverse feature sets and the stability of its predictions across various conditions underscore its relevance in this study, positioning it as a viable alternative for comparison with more sophisticated models.

4.3 Neural Networks

Neural networks are a class of machine learning models inspired by the biological neural networks that constitute animal brains. These models are designed to recognise patterns in data through a structure of interconnected nodes or neurons. Each neuron receives input, processes it, and passes its output to subsequent neurons, forming a complex web of relationships. Mathematically, the output of a neuron can be expressed as $y = f(W^T x + b)$, where x represents the input vector, W denotes the weights, b is the bias, and f is an activation function that introduces non-linearity. Neural networks are particularly effective in capturing complex, non-linear relationships within data, making them suitable for a wide range of tasks, including classification, regression, and time series forecasting, such as predicting recipe rank percentiles. The architecture of neural networks typically consists of an input layer, one or more hidden layers, and an output layer, with various activation functions used to introduce non-linearity into the model [Goodfellow et al., 2016].

In this study, both the Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) models are compiled using the `adam` optimiser [Kingma and Ba, 2014], which adapts the learning rate during training, and the Mean Absolute Error (MAE) loss function, providing an interpretable measure of error.

4.3.1 Recurrent Neural Network (RNN)

The Recurrent Neural Network (RNN) is a type of neural network specifically designed to process sequences of data. RNNs maintain a hidden state that is updated at each time step, allowing the model to capture sequential dependencies in the input data [Elman, 1990]. The hidden state h_t at time step t is calculated as follows:

$$h_t = f(W_h \cdot h_{t-1} + W_x \cdot x_t + b)$$

where f is an activation function, W_h is the weight matrix for the hidden state, W_x is the weight matrix for the input features x_t, and b is a bias term. This formulation enables RNNs to effectively process sequential information, making them particularly suitable for time series forecasting tasks, where the temporal order of the data is crucial.

Architecture

The architecture of this study's RNN includes the following components:

- **Input Layer**: The input shape is $(\text{time\_steps}, \text{features})$, where each time step contains the feature vector for that moment in time.
- **RNN Layer 1**: The first RNN layer consists of a specified number of units. The `return_sequences=True` parameter ensures that the entire sequence is passed to the next layer, allowing the network to capture patterns across all time steps.

- **Dropout Layer 1**: A 20% dropout layer is applied after the first RNN layer to prevent overfitting by randomly dropping units during training.
- **RNN Layer 2**: A second RNN layer with the same number of units is added, with `return_sequences=False`, as only the final hidden state is needed for the output.
- **Dropout Layer 2**: Another dropout layer, also with a 20% dropout rate, is included after the second RNN layer to further mitigate overfitting.
- **Dense Layer**: A fully connected dense layer with a single output unit produces the final predicted value, such as the recipe rank percentile.

RNNs are particularly effective for capturing short-term sequential dependencies, as they are designed to leverage the information from recent data points more heavily. However, they often struggle with long-term dependencies due to the vanishing gradient problem [Goodfellow et al., 2016]. This issue occurs when gradients diminish as they are backpropagated through many time steps, making it challenging for the model to learn long-range correlations within the data.

4.3.2 Long Short-Term Memory (LSTM)

The *Long Short-Term Memory (LSTM)* network is an advanced variant of RNNs, designed to effectively manage longer-term dependencies through its internal memory cells [Hochreiter and Schmidhuber, 1997]. Unlike standard RNNs, LSTMs incorporate gating mechanisms to control the flow of information, via the input gate i_t, forget gate f_t, and output gate o_t. Mathematically, LSTMs utilise the Hadamard product [Horn and Johnson, 2012], which denotes element-wise multiplication, enabling selective retention or updating of the cell state c_t via equations such as:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t,$$

where f_t and i_t are the outputs of the forget and input gates, respectively, and $\tilde{c}_t$ represents the candidate cell state [Greff et al., 2017]. These mechanisms allow LSTMs to avoid the vanishing gradient problem, facilitating the retention of relevant information from both recent and distant time steps.

Architecture:

The architecture of this study's LSTM includes the following components:

- **Input Layer**: The input shape is $(\text{time\_steps}, \text{features})$, identical to that of the RNN.
- **LSTM Layer 1**: The first LSTM layer consists of a specified number of units and incorporates gating mechanisms to control the flow of information. The `return_sequences=True` parameter ensures that outputs from all time steps are passed to the next layer.
- **Dropout Layer 1**: A 20% dropout layer is applied after the first LSTM layer to reduce overfitting.
- **LSTM Layer 2**: A second LSTM layer with the same number of units is added, with `return_sequences=False`, as only the final hidden state is needed for the output.

- **Dropout Layer 2**: A second dropout layer, again with a 20% dropout rate, is applied after the second LSTM layer.
- **Dense Layer**: The final dense layer consists of a single output unit, corresponding to the predicted value.

The LSTM architecture is designed to maintain relevant information from both recent and distant time steps, making it effective for tasks where the target value relies on observations from multiple time steps in the past. Its internal gating mechanisms, including input, forget, and output gates, regulate the flow of information within the memory cell, allowing for better management than standard RNNs. This structured approach helps mitigate the vanishing gradient problem and enhances performance in applications requiring the learning of extended sequences and complex patterns.

In summary, while RNNs capture short-term dependencies, LSTMs excel in scenarios where long-term context is crucial for accurate predictions. The architectural differences, such as the presence of memory cells and specific gating mechanisms, significantly improve their ability to model temporal relationships in data, making LSTMs a preferred choice for time series forecasting tasks.

4.4 Ranking Models

Ranking models are designed to predict the relative order of items, rather than their absolute values. These models take into account the relationships between recipes within the same menu week, rather than generating independent pointwise predictions for each recipe. By learning to rank recipes relative to one another in the dataset D, ranking models optimise specific ranking metrics, such as the Normalised Discounted Cumulative Gain (NDCG), which is used in this study as the evaluation metric for all ranking models.

The Normalised Discounted Cumulative Gain (NDCG) is an evaluation metric that measures the quality of ranked lists by emphasising the importance of correctly ranking the most relevant items at the top of the list. For a set of k items, NDCG is defined as:

$$\mathrm{NDCG@k} = \frac{\mathrm{DCG@k}}{\mathrm{IDCG@k}},$$

where DCG@k (Discounted Cumulative Gain) is:

$$\mathrm{DCG@k} = \sum_{i=1}^{k} \frac{2^{y_i} - 1}{\log_2(i+1)},$$

and IDCG@k represents the ideal ordering of items in the top k positions. The NDCG metric is used to evaluate the performance of all ranking models discussed in this section.

4.4.1 LightGBM Ranker (LambdaMART)

The *LightGBM Ranker* employs the LambdaRank algorithm, which is a ranking-specific extension of gradient-boosted decision trees. LambdaRank optimises ranking performance by adjusting the model based on pairwise comparisons between items and their contribution to the NDCG score [Burges, 2010].

In this study, LightGBM's built-in LambdaRank loss function is used as the objective during training, with NDCG as the evaluation metric. The LambdaRank loss optimises ranking order by comparing the predicted scores of items i and j within a group, and adjusting the model to maximise the NDCG score. The loss function is defined as:

$$\mathcal{L}_{\text{LambdaRank}} = \sum_{i \in D} \sum_{j \in D} \lambda_{ij} \cdot \left(-\sigma(z_{ij}) \cdot \lambda_{ij}\right),$$

where $z_{ij} = \hat{y}_i - \hat{y}_j$ represents the pairwise difference between predicted scores $\hat{y}_i$ and $\hat{y}_j$, and λ_{ij} reflects the effect on NDCG if the ranks of items i and j were swapped. Specifically,

$$\lambda_{ij} = \text{NDCG}(rank(j)) - \text{NDCG}(rank(i)).$$

Here, $\sigma(z_{ij})$ is the sigmoid function, ensuring that the model adjusts the predicted scores smoothly to reduce ranking errors, while the λ_{ij} terms guide the optimisation towards better ranking performance in terms of NDCG.

Although NDCG is computationally expensive to optimise directly, the LambdaRank approach approximates improvements in NDCG by calculating these lambdas. The model then iteratively adjusts the predicted scores to minimise this loss function, thereby improving the NDCG score upon evaluation.

LambdaMART, another popular ranking algorithm used in conjunction with gradient boosting, builds on the same principles as LambdaRank but applies them within the framework of multiple decision trees [Burges, 2010]. Each decision tree is trained on the residuals of the previous trees, which represent the lambdas computed during training. The combined model learns to predict scores that optimise NDCG for ranking tasks. In this study, while the LightGBM Ranker was trained using the LambdaRank loss function, the process shares similarities with LambdaMART in its use of gradient-boosted trees and ranking-specific objectives [Ke et al., 2017].

By setting k to the size of the menu week in each group (up to 155), it is ensured that the model captured relationships between all recipes within a week, rather than focusing only on the top few items. This configuration allows for more accurate prediction of recipe ranks, ensuring that all items are evaluated on the same scale.

Overall, the LightGBM Ranker, with its LambdaRank objective and the NDCG evaluation metric, is well-suited to tasks like recipe ranking, where relative ordering between items is critical.

4.4.2 XGBoost Ranker

The *XGBoost Ranker* is a powerful gradient boosting model that excels in both classification and ranking tasks [Liu, 2009]. It is particularly well-suited for learning-to-rank problems, where the goal is to predict the relative ordering of items within a group. XGBoost Ranker uses a highly efficient implementation of gradient boosting, which makes it scalable and effective for large datasets, such as ranking recipes within menu weeks.

XGBoost Ranker is trained using a pairwise ranking objective, known as `rank:pairwise`. This objective focuses on learning the relative ordering between pairs of items within the same group, ensuring that recipes with higher true rankings are predicted to have higher scores. This approach is especially useful in scenarios where the ranking order of items is more important than their absolute values.

The pairwise ranking loss function optimised by XGBoost Ranker is:

$$\mathcal{L}_{\text{pairwise}} = \sum_{i \in D} \sum_{j \in D} \mathbf{1}(y_i > y_j) \cdot \log(1 + \exp(\hat{y}_j - \hat{y}_i)),$$

where y_i and y_j are the true rank percentiles for recipes i and j in the dataset D, and $\hat{y}_i$ and $\hat{y}_j$ are their respective predicted values. The indicator function $\mathbf{1}(y_i > y_j)$ ensures that the loss is only calculated for pairs where recipe i should be ranked higher than recipe j. This objective adjusts the predicted margin between recipes i and j, with a greater penalty applied when the model's predictions incorrectly rank lower-ranked recipes higher.

In addition to the pairwise loss, XGBoost leverages a second-order Taylor expansion to approximate the objective function. This allows the model to efficiently compute both the loss and its approximation to improve the training process. The model's optimisation is done by minimising the following second-order approximation of the loss:

$$\mathcal{L}(\hat{y}) \approx \mathcal{L}(\hat{y}_0) + \nabla_{\hat{y}} \mathcal{L}(\hat{y}_0)(\hat{y} - \hat{y}_0) + \frac{1}{2} \nabla^2_{\hat{y}} \mathcal{L}(\hat{y}_0)(\hat{y} - \hat{y}_0)^2,$$

where $\nabla_{\hat{y}} \mathcal{L}$ and $\nabla^2_{\hat{y}} \mathcal{L}$ represent the first and second-order gradients (Hessian) of the loss function with respect to the predicted values $\hat{y}$. This approximation allows XGBoost to update its predictions efficiently by considering both the gradient and curvature of the loss function.

In recipe ranking tasks, where multiple recipes within a menu week compete for higher rankings, the pairwise approach effectively captures these interdependencies. XGBoost Ranker, by comparing pairs of recipes within the same group, adjusts the ranking in a way that aligns with the true order of recipes in terms of their relevance or popularity.

Although the pairwise ranking objective is used for training, the evaluation metric in this study is Normalised Discounted Cumulative Gain (NDCG). Again, k is set to the maximum group size since the final evaluation metric of MAE weights all recipes equally.

XGBoost's ability to handle large datasets efficiently, combined with the pairwise ranking objective and its second-order optimisation, makes it particularly well-suited for tasks where the relative ordering of recipes is critical.

4.4.3 CatBoost Ranker

The *CatBoost Ranker* is a gradient boosting model specifically designed to handle categorical features efficiently [Prokhorenkova et al., 2018]. Like other ranking models, CatBoost Ranker is used to predict the relative order of items within groups, such as recipes within a menu week. Its specialised pairwise ranking approach enables the model to focus on the relationships between items in a group, making it effective for tasks where ranking order is critical.

CatBoost Ranker uses the `PairLogit` loss function, which optimises the pairwise classification between items. This loss function ensures that recipes with higher true rankings receive higher predicted scores, providing an effective solution for ranking tasks where the goal is to preserve the relative ordering of items within a group.

The `PairLogit` loss function optimised by CatBoost Ranker is defined as:

$$\mathcal{L}_{\text{PairLogit}} = \sum_{i,j \in D} \log(1 + \exp(-(y_i - y_j)(\hat{y}_i - \hat{y}_j))),$$

where y_i and y_j are the true rank percentiles for recipes i and j in the dataset D, and $\hat{y}_i$ and $\hat{y}_j$ are their respective predicted values. This loss function minimises the error by adjusting the predicted margins so that higher-ranked recipes are correctly scored above lower-ranked ones.

CatBoost's pairwise ranking also benefits from the model's ability to efficiently handle categorical features. This applies well within this study, where many core features are categorial (ordinal) e.g. `month`, `new_recipe`, `num_variants_on_menu`. During training, categorical variables are transformed into numerical values using a technique called ordered boosting, which prevents overfitting and leads to more accurate rankings. The transformation relies on the target statistics of the categorical variables, incorporating the following formula:

$$\text{CatFeature}_i = \frac{\sum_{t=1}^{T_i} y_t}{T_i + \alpha},$$

where T_i is the number of past observations for the categorical feature i, y_t is the target value of the t-th observation, and α is a smoothing parameter that controls the influence of each new observation on the feature transformation. This method allows CatBoost Ranker to handle high-cardinality categorical features efficiently without the risk of overfitting.

Like the other rankers, the CatBoost Ranker is evaluated using the Normalised Discounted Cumulative Gain (NDCG) metric.. Once again, k is set to the maximum group size since the final evaluation metric of MAE weights all recipes equally.

CatBoost's efficient handling of categorical features and its robust pairwise ranking mechanism make it well-suited for tasks like recipe ranking, where the relationships between recipes within a group must be captured effectively to ensure accurate predictions.

4.5 Models Summary

Section 4 provided a comprehensive overview of all models for use within this study, detailing algorithms, loss functions and suitability to a ranking task.

The main aim will be to compare performance across multiple models and identify the most suitable model, with considering both MAE and the calculated confidence intervals. In particular, a hypothesis test between the best performing pointwise and ranking models may provide strong insights into the suitability of learning-to-rank algorithms with the study's ranking task.

Chapter 5

Experiments

This section detailed the experiments conducted and engineering design used throughout the building of both long-term and short-term models.

5.1 Computational Resources

The experiments were implemented using Python 3.12.7, which provides a comprehensive ecosystem for machine learning and data analysis. The following Python libraries and packages were utilised for various stages of model development, training, and evaluation:

- **Pandas** and **NumPy**: Used for data manipulation, preprocessing, and handling large datasets.
- **Matplotlib** and **Seaborn**: Used for visualising data, plotting metrics, and generating performance graphs.
- **Scikit-learn**: Utilised for data scaling (StandardScaler, MinMaxScaler), PCA (Principal Component Analysis), model evaluation (Mean Absolute Error), and training models like Linear Regression and Random Forest Regressor.
- **Optuna**: Used for hyperparameter tuning, including visualisation tools from `optuna.visualization`.
- **XGBoost**, **LightGBM**, and **CatBoost**: Gradient-boosting libraries used for building and training ranking models.
- **TensorFlow** and **Keras**: Used for building and training deep learning models such as RNNs and LSTMs.
- **OS** and **JSON**: Used for file handling and configuration management.

The experiments were performed on a high-performance personal computer with the following hardware specifications:

- **Processor (CPU)**: AMD Ryzen 5 3600 6-Core Processor.
- **Graphics Processing Unit (GPU)**: NVIDIA GeForce RTX 2060 with 6GB VRAM.

- **RAM**: 32GB DDR4, which enabled efficient handling of large datasets and memory-intensive operations.
- **Storage**: 1TB NVMe SSD, providing fast read/write operations, however the dataset size of 282MB does not require significant storage capabilities.

By utilising the combination of CPU and GPU resources, the training and evaluation processes were optimised, ensuring timely completion of experiments even with large datasets and complex models.

5.2 Data Pre-processing

The binary feature `contamination_outlier`, which indicates the uniqueness or novelty of a recipe compared to others, was removed because this information is not available in the test data. Additionally, the feature `uptake_at_lead_day_-6`, representing uptake observed on `lead_day_7`, was excluded. This feature would provide identical predictions to `lead_day_k` for $k \in \{4, 5, 6\}$, and holds no value since the final delivery date precedes this point, with the delivery window for the new menu week already starting by then.

5.2.1 Data Normalisation

Two distinct scaling methods were applied to the features in the long-term dataset, depending on the requirements of different models.

First, the `StandardScaler` was utilised to ensure that the features have a mean of zero and a standard deviation of one. This step is crucial for models such as Linear Regression, which assume normally distributed input features on a comparable scale. Without normalisation, features with larger scales could dominate the learning process, leading to biased results and suboptimal model performance.

For recurrent neural networks (RNNs) and LSTM neural networks, the dataset was scaled using the `MinMaxScaler`, which transforms the features into a range between 0 and 1. This scaling is critical for neural networks, as they perform more effectively when input features are within a standardised range, mitigating the risks of gradient-related issues such as vanishing or exploding gradients [Goodfellow et al., 2016]. MinMax scaling also ensures that all features are treated equally by the activation functions used in RNNs and LSTMs, which are sensitive to input magnitudes.

5.2.2 Rank Percentile Generation

The transformation of the target variable from `recipe_uptake` to `recipe_rank_percentile` required careful handling and data manipulation. To first compute the `recipe_rank`, uptakes were grouped by `menu_week` and ranked within each week in descending order. This process assigns an ordinal rank of 1 to the highest uptake, while the lowest uptake receives a rank of n, where n is the number of recipes in that particular week.

Before converting ranks into percentiles, it is important to note that certain ranking models, such as LightGBM Ranker and XGBoost Ranker, require the use of raw ordinal ranks. These models rely on pairwise ranking objectives, where items are compared within the same group to optimise their ranking order. To ensure proper interpretation by these models, the ordinal ranks are adjusted to a range of 0 to $n-1$ by subtracting 1 from all ranks. This adjustment ensures that the models correctly interpret the relative positioning of items for pairwise comparisons [Developers, 2024].

Although CatBoost Ranker also utilises a pairwise ranking objective (via the PairLogit function), it is more flexible in handling target ranks and can work effectively with both raw ordinal ranks and continuous values like rank percentiles, hence this study continue with rank percentiles for CatBoost Ranker.

Initial analysis revealed that recipe ties in uptake were rare, occurring in only 5 menu weeks, making ties negligible in this dataset. For consistency, ties are handled by assigning the higher rank to the recipe that appears first in the dataset, using the `pandas` rank function with `'method'=first`. Given the low frequency of ties and the large number of recipes in some menu weeks, this approach has a minimal impact on the results.

After generating ordinal ranks, they are transformed into percentiles, so the target is evenly distributed between 0 and 1, with 1 representing the highest rank and uptake. The rank percentile is used as the target for all models except LightGBM Ranker and XGBoost Ranker. However, all models, regardless of training target, are evaluated on rank percentiles. This means that all predictions are converted to rank percentiles within the test week, and the training data is either converted into rank percentiles before or after training, depending on the model in use.

5.3 Evaluation Framework Setup

The Multi-Week Forecasting Framework for the long-term model, described in Section 3.3.2, ensures proper handling of time-series data, which is indexed by menu week. The long-term model is evaluated from 1 to 8 weeks prior to the target week, iterating over 52 target menu weeks to provide a comprehensive view of a year's performance while accounting for seasonality. Given that the dataset contains menu weeks from 109 to 619, testing 52 weeks represents approximately 10% of the dataset, ensuring sufficient training data for effective modelling.

Although the short-term model's results ultimately surpass the long-term model within the last 18 days (or 3 weeks), evaluating the long-term model across the entire 1 to 8 week domain is still essential. This provides a broader view of the long-term model's performance and justifies the need for a short-term model, given the performance improvement observed when both models are combined.

The evaluation framework trains on menu weeks from 109 to $N-k$, with the test set being the single menu week N. Here, N is initially set to the most recent menu week, i.e., week 619. This process fixes the test week, and trains $K=8$ models, each with a different maximum menu week in the training data, where $k \in \{1, 2, \cdots, K\}$ represents the training lag behind the test week. Once 8 models are trained for test week N, the procedure repeats for test week $N-1$, training up to $N-k-1$ using 8 models with varying k values.

This process continues until the 52 most recent menu weeks have been evaluated with 8 models each. The MAE values for each k are averaged to compute the mean MAE for each $k \in \{1, 2, \cdots, 8\}$ weeks before the target. A 95% confidence interval, calculated via the standard error, is plotted as a shaded region to reflect the uncertainty in each mean MAE value. This enables a view of each model's lagged performance and the improvements from 8 weeks to 1 week before the target.

The algorithm below summarises the approach, detailing the training loops and structure used:

Algorithm 1 Multi-Week Forecasting Framework Algorithm

```
Input: Time-series data (menu weeks 109 to 619), maximum lag $K = 8$, $T = 52$ test weeks
Output: Mean MAE for each $k \in \{1, 2, \cdots, K\}$, 95% confidence interval
Initialise:
 Set $N = 619$ (most recent menu week)
 Set $K = 8$ (maximum lag)
 Set $T = 52$ (number of test weeks)

while $N \geq 619 - T + 1$ do
    for $k \in \{1, 2, \cdots, K\}$ do
        Train model on menu weeks 109 to $N - k$
        Evaluate model on test week $N$
        Record MAE for $k$
    end
    Decrement $N$ by 1
end
for $k \in \{1, 2, \cdots, K\}$ do
    Compute mean MAE across all test weeks
    Calculate 95% confidence interval using standard error
end
```

This multi-week algorithm trains $T \times K$ (52×8) total models, where $K = \max(k)$ is the largest lag evaluated. Each model is trained on up to 27,330 recipes and 214 features, repeated across 11 predictive model types as detailed in Chapter 4. This emphasises the importance of optimising total training and evaluation time throughout the study, while still providing a robust and intuitive approach to conducting the proposed experiments.

5.3.1 Inference Pipeline Optimisation

The multi-week evaluation framework described in Section 5.3 can be further optimised to significantly reduce total training and evaluation times by altering the order in which training and inference are performed. This optimisation is possible because the same trained model can be used for multiple inferences, eliminating the need for duplicate models.

This study identifies an efficient approach based on the assumption that inference times are negligible compared to training times. The new strategy performs the same number of inferences but reduces the number of training calls by first fixing the menu weeks for each trained model and then performing the required inferences across multiple test weeks. This contrasts with the previous approach, where the test week was fixed, and models were trained for each relevant set of menu weeks.

For example, instead of testing on week 610 and training 8 models, each using weeks 602-609, a model is first trained up to week 602 and evaluated on test weeks 603-610 (These are not equivalent operations, however 8 evaluations are made in both cases but only 1 model is trained in the latter, with 8 in the former). This exhausts all possible inferences for that trained model. In general, models are trained on weeks 109 to M, then used to predict weeks $M + k$ with $k \in \{1, 2, \cdots, K\}$, with $K = 8$ and ensuring that $M + k \leq 619$. This optimisation reduces the number of models trained from $T \times K$ to $T + K - 1$. In this study, the total number of models is reduced from 416 to 59, resulting in an 85.8% reduction in training time, assuming equal training time per model.

The following algorithm outlines the optimised inference pipeline:

Algorithm 2 Optimised Inference Pipeline for Multi-Week Forecasting Framework

```
Input: Time-series data (menu weeks 109 to 619), lag range $k \in \{1, 2, \cdots, K\}$, $T = 52$ test weeks
Output: Mean MAE for each $k$, 95% confidence interval
Initialise:
 Set $N = 619$ (most recent menu week)
 Set $K = 8$ (maximum lag)
 Set $T = 52$ (number of test weeks)
 Set $M = N - T - K + 1$ (initial training menu week)

while $M \leq N - 1$ do
    Train model on menu weeks 109 to $M$
    for $k \in \{1, 2, \cdots, K\}$ such that $M + k \leq N$ do
        Evaluate model on test week $M + k$
        Record MAE for $k$
    end
    Increment $M$ by 1
end
for $k \in \{1, 2, \cdots, K\}$ do
    Compute mean MAE across all test weeks
    Calculate 95% confidence interval using standard error
end
```

This approach requires each model to be retained temporarily between inferences rather than being replaced after each one, as occurs without this optimisation. However, the statistical models discussed in Section 4.1 do not have callable functions for separate inference, as training and inference are conducted simultaneously. As such, this optimisation does not apply to these models, but due to their short evaluation times, this issue is negligible, and they can follow the unoptimised algorithm.

It is also important to note that the above algorithms do not mention the training window sizes applied to each model, as described in Sections 3.3.2 and 5.4.2. This omission does not affect the algorithms or the optimisation since each model is trained on the most recent W menu weeks, where W is the window size chosen through tuning for each model type. This would just change each model to train on menu weeks $M - \max(109, W)$ to M.

5.4 Optuna Hyperparameter Tuning

Optuna was selected to perform hyperparameter tuning for each model, with 200 trials allocated to find the optimal hyperparameters that minimise the mean MAE via 3-fold time series cross-validation. Hyperparameter tuning was conducted for LightGBM, XGBoost, Random Forest, LightGBM Ranker, XGBoost Ranker, and CatBoost Ranker, focusing on several key parameters, including learning rate, loss function, and evaluation metric.

For the long-term model, the best-performing hyperparameters for the supervised learning models are listed below:

Hyperparameter	**LightGBM**	**XGBoost**
`objective`	regression	reg:logistic
`metric/eval_metric`	mae	mae
`learning_rate`	0.2200	0.1759
`max_depth`	18	18
`num_leaves/min_child_weight`	669	2
`min_child_samples`	59	-
`min_split_gain/gamma`	1.467e-06	0.06299
`subsample`	0.7383	0.8156
`colsample_bytree`	0.9649	0.5720
`colsample_bylevel`	-	0.8214
`lambda_l1/lambda`	4.840e-07	0.06734
`lambda_l2/alpha`	0.00256	0.8232

Table 5.1: Optuna-tuned Hyperparameters for LightGBM and XGBoost

During the tuning process, both predictions and actual values were converted to rank percentiles, and the MAE was used as the final evaluation metric, similar to Section 5.2.2.

The optimal hyperparameters for the three ranking models are also displayed below, where pairwise objective functions were preferred over pointwise objectives. However, tuning took significantly longer, as pointwise ranking has a time complexity of $O(n)$, while pairwise ranking has a time complexity of $O(n^2/g)$, where n is the number of items, and g is the number of groups [Kannen et al., 2024].

Hyperparameter	**LightGBM Ranker**	**XGBoost Ranker**	**CatBoost Ranker**
`objective`	lambdarank	rank:pairwise	PairLogit
`eval_metric`	NDCG	NDCG	NDCG
`lr`	0.0673	0.1414	0.3477
`max_depth`	13	6	9
`min_child_weight`	6.21	17	-
`min_child_samples`	51	-	-
`gamma/split_gain`	-	1.05	-
`subsample`	0.77	0.93	0.94
`subsample_freq`	8	-	-
`colsample_bytree`	0.70	0.66	-
`colsample_bylevel`	-	0.51	0.79
`lambda_l2/lambda`	0.003	0.009	-
`reg_alpha/alpha`	0.001	0.23	-
`l2_leaf_reg`	-	-	9.36
`bootstrap_type`	-	-	Bernoulli
`importance_type`	split	-	-
`label_gain/ndcg_exp_gain`	range(max(group))	False	-

Table 5.2: Optuna-tuned Hyperparameters for LightGBM Ranker, XGBoost Ranker, and CatBoost Ranker

For neural networks (RNN and LSTM), hyperparameters were tuned using a grid search approach, optimising for units, batch size, and number of epochs. The following hyperparameters were obtained for both models:

Parameter	**RNN Model**	**LSTM Model**
`Units`	64	64
`Dropout Rate`	0.2 after each layer	0.2 after each layer
`Optimiser`	Adam	Adam
`Loss Function`	Mean Absolute Error	Mean Absolute Error
`Epochs`	300	200
`Batch Size`	16	8

Table 5.3: Grid-Search tuned Hyperparameters for RNN and LSTM Models

5.4.1 Custom Time Series Cross-validation

The validation dataset and approach used for hyperparameter tuning involve a custom time-series 3-fold cross-validation method. This method selects three menu weeks as validation weeks, ensuring that they occur before any menu weeks used in the final test set. From the multi-week evaluation framework in Section 3.3.2, the earliest week used for testing is menu week $568 = 619 - 52 + 1$. Therefore, the validation set consists of menu weeks 564 to 567. Training is conducted on all menu weeks preceding each validation week, with a time lag of only one week at each step [Bergmeir and Benítez, 2012]. The MAEs for the three validation weeks are averaged, yielding the trial performance of the chosen hyperparameters and model.

This custom approach was necessary to appropriately handle menu weeks, rather than simply selecting the last three recipes. As a result, the validation dataset was extracted at the week level, rather than at the recipe level. The standard `TimeSeriesSplit` function from `scikit-learn` does not allow for the proper grouping of recipes by menu week, necessitating this customised solution.

5.4.2 Training Window Size Tuning

Introducing a training window for each model's evaluation can enhance performance and reduce total training time [Brownlee, 2017]. A training window of size W allows a model to train using only the most recent W menu weeks, rather than all menu weeks from the start (week 109). This approach is especially useful in production, where the focus is on predicting recent menu weeks, since training on data from over 500 weeks ago (roughly 10 years) may not best represent current menu trends. Although the models, indexed by menu week, may naturally weigh recent weeks more heavily, a training window can further reduce such noise and irrelevant data.

Moreover, early menu weeks had significantly fewer recipes per week (around 10), while recent weeks contain approximately 120 recipes. Predicting on early weeks with 10 recipes could present a different challenge compared to weeks with over 100 recipes, potentially requiring different modelling approaches. Figure 5.1 illustrates the gradual increase in the number of recipes per menu week over time.

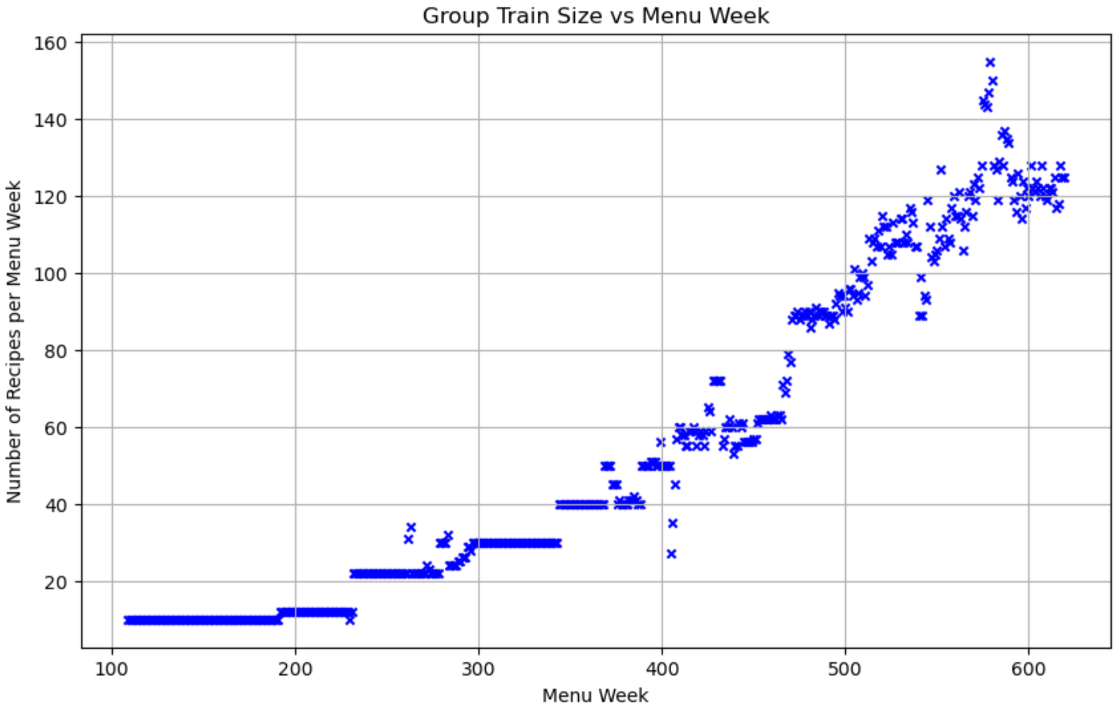

Figure 5.1: Scatter plot of the Menu Week index and the corresponding number of recipes within each week.

This presents a trade-off: models must balance between using more data (recipes and menu weeks) and focusing on similar-sized menu weeks with smaller datasets. Since the models predict rank percentiles, larger menu weeks require greater precision, as the target values become more condensed. However, older menu weeks might still contain useful recipe information, aiding in pointwise evaluations.

The training window sizes were tuned for each model, and the results are shown in Figure 5.2. A grid search was conducted in intervals of 50, optimising for all models.

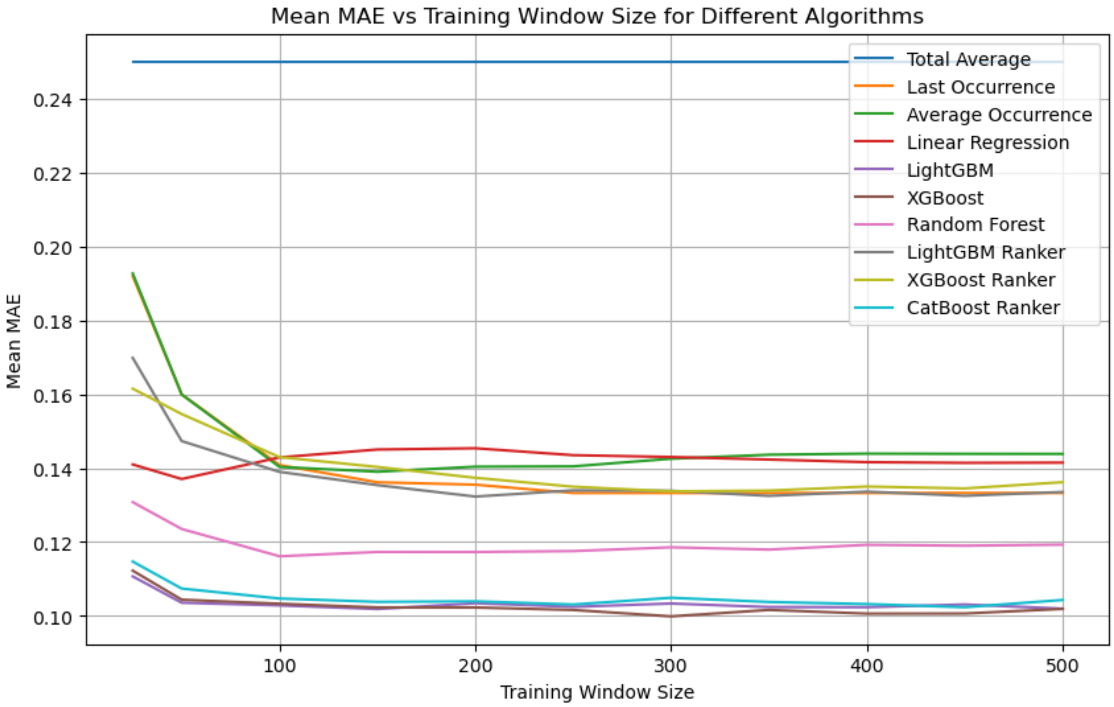

Figure 5.2: Comparison of training window sizes for various statistical, pointwise, and ranking models.

Neural networks were trained without a window size, indicated as `None`, meaning all available data was used. The final hyperparameter choices for window size are presented in Table 5.4, considering both mean MAE performance and training times.

Model	**Training Window Size**
Total Average Algorithm	None
Last Occurrence Algorithm	None
Average Occurrence Algorithm	120
Linear Regression	50
LightGBM	100
XGBoost	300
Random Forest	120
RNN	None
LSTM	None
LightGBM Ranker	180
XGBoost Ranker	300
CatBoost Ranker	250

Table 5.4: Optimal Training Window Size for Each Model via Grid-Search

5.5 Real-Time Predictions Setup

This section details the adaptations of the long-term model setup for the short-term model, where real-time data becomes the pivotal feature set for predictions within the 18 days leading up to the first delivery date. Concepts such as training windows and inference pipeline optimisation, discussed for the long-term model, are not applicable here since short-term features are only available starting from menu week 488 onwards.

5.5.1 Real-Time Feature Generation

The real-time data from customers' live recipe selections, `uptake_at_lead_day_n` for $n \in \{18, 17, \cdots, -4, -5\}$, undergoes transformations to generate additional features, as described in Section 3.5.1. First, the uptakes are converted directly into rank percentiles for all models. Next, a new set of 23 features, `diff_uptake_at_lead_day_n`, is created to represent the difference in rank percentile between consecutive lead days, capturing the change between `uptake_at_lead_day_n+1` and `uptake_at_lead_day_n`.

For example, in the multi-day forecasting framework (Section 3.5.2), assume the model aims to predict the target on `lead_day_15`. In this case, real-time uptake and difference data is available for $n \in \{18, 17, 16\}$. The model will use the latest uptake value (`uptake_at_lead_day_16`) and all available difference values (`diff_uptake_at_lead_day_n` for $n \in \{17, 16\}$) to capture the change in target, effectively tracking the gradient from one day to the next. The uptake values operate like a sliding window (of size 1) in the training dataset as predictions approach the target, while the difference values form an expanding window, retaining older data points.

Excluding older uptakes reduces potential noise during model training, as including all features may cause the model to overemphasise them. Older uptakes generally offer little additional information beyond what the most recent uptake already provides, whereas the change in uptake (captured by `diff_uptake_at_lead_day_n`) remains critical. Correlation analysis (Section 3.2.1) has shown that the real-time uptake features exhibit continuous improvement in their relationship with the target as predictions near the delivery date. Therefore, incorporating older uptake values might hinder the model's ability to leverage other non-uptake features in the dataset, potentially leading to an over-reliance on uptake data.

5.5.2 Multi-Day Forecasting Framework Setup

The evaluation framework for the short-term model, described in Section 3.5.2, allows for the visualisation of model performance as long-term features are incrementally combined with short-term features, as outlined in Section 5.5.1. By aggregating model performance over 52 iterations, the framework provides an effective analysis of the model's predictive ability at any given `lead_day`.

The algorithm for the multi-day framework is similar to Algorithm 1 and is described below:

Algorithm 3 Algorithm for the Multi-Day Forecasting Framework for Short-Term Model

```
Input: Long-term data (menu weeks 488 to 619), short-term features, maximum day lag $K = 21$,
       $T = 52$ test weeks
Output: Mean MAE for each $k \in \{-6, -5, \cdots, K\}$, 95% confidence intervals
Initialise:
  Set $N = 619$ (most recent menu week)
  Set $K = 21$ (maximum lag for short-term model)
  Set $T = 52$ (number of test weeks)

while $N \geq 619 - T + 1$ do
    for $k \in \{-6, -5, \cdots, K\}$ do
        Real-Time Feature Inclusion:
        if $k \leq 16$ then
            Include uptake_at_lead_day_{k+1} column (real-time uptake data)
            for $j \in \{16, 15, \cdots, k\}$ do
                Include diff_uptake_at_lead_day_{j+1} columns (real-time difference data)
            end
        end
        if $k = 17$ then
            Include only uptake_at_lead_day_{k+1} (no difference data)
        end
        if $k > 17$ then
            Use only long-term features (no real-time data)
        end
        Buffer Calculation:
          Define buffer $b = \left\lfloor \frac{\max(0,k+2)}{7} \right\rfloor + 1$

        Training:
          Train model on menu weeks 488 to $N - b$ with appended real-time and long-term features

        Evaluation:
          Evaluate model on test week $N$
          Record MAE for each $k$
    end
    Decrement $N$ by 1
end
for $k \in \{-6, -5, \cdots, K\}$ do
    Compute mean MAE across all test weeks
    Calculate 95% confidence interval using standard error
end
```

Each model is compared against the short-term baseline predictive model, which uses `uptake_at_lead_day_n` (transformed into a rank percentile) as the prediction value. The exploratory data analysis conducted in Section 3.2.2 indicates that this baseline model would still serve as a strong predictor due to the high correlations and eventual convergence of uptake values, providing a reference point to evaluate how much incorporating long-term features and difference feature generation improves performance.

The inference pipeline optimisation techniques described in Section 5.3.1 were not feasible within the short-term model framework. Since both rows and columns are independently concatenated to the training dataset depending on whether short-term uptake features or recipes from new menu weeks are included, this prevents reusing a trained model for multiple inferences. While this increases bulk training times, it will have minimal impact on isolated inferences for new menu weeks within production, as the same number of models will be trained per task.

5.6 Hypothesis Test Setup

This section describes the framework for conducting the hypothesis tests in this study, focusing on the statistical methods used to evaluate model performance. The tests aim to determine whether there is a statistically significant difference in Mean Absolute Error (MAE) between models across both long-term and short-term forecasting tasks. The significance level for all tests is set at $\alpha = 0.05$, meaning that if the p-value is less than 0.05, the null hypothesis is rejected in favor of the alternative hypothesis.

5.6.1 T-statistic and P-value Calculation

The hypothesis tests compare the MAE distributions of two models across 52 menu week iterations. A paired two-tailed t-test is used to determine if there is a significant difference in MAE between the two models.

The t-statistic is calculated as:

$$t = \frac{\bar{d}}{s_d/\sqrt{n}}$$

Where:

- $\bar{d}$ is the mean difference in MAE between the two models.
- s_d is the standard deviation of the differences.
- $n = 52$ is the number of menu week iterations.

The corresponding p-value is calculated as:

$$p = 2 \cdot P(T > |t|)$$

Where T follows a t-distribution with $n - 1$ degrees of freedom. If the p-value is less than 0.05, the null hypothesis is rejected, indicating that there is a significant disparity in performance between the models.

5.6.2 Confidence Interval Calculation

A two-tailed 95% confidence interval for the difference in MAE is calculated using:

$$\mathrm{CI} = \left(\bar{d} - t_{\alpha/2,n-1} \cdot \frac{s_d}{\sqrt{n}}, \bar{d} + t_{\alpha/2,n-1} \cdot \frac{s_d}{\sqrt{n}}\right)$$

Where $t_{\alpha/2,n-1}$ is the critical value of the t-distribution at $\alpha = 0.05$ and $n - 1 = 51$ degrees of freedom: for this study, $t_{0.025,51} = 2.0076$. This interval helps quantify the plausible range of differences in MAE between the models.

5.6.3 Summary and Interpretation

The results of the hypothesis tests are interpreted based on the p-value and confidence interval. If the p-value is below the significance level of 0.05, a conclusion is drawn that the models have significantly different MAEs. The confidence interval provides further context by estimating the range within which the true difference in MAE lies.

By employing this approach, the study evaluates whether the observed differences in model performance are statistically significant, offering insights into the comparative effectiveness of the tested models. The sign of the t-statistic will help determine which model performs better.

Chapter 6

Results & Analysis

This chapter displays and analyses the final model results, attained from the methodology, models and experiments described within prior sections. This is split into long-term and short-term performance per model, and further split into model family for a complete comparative analysis. The chapter concludes with hypothesis tests which quantify uncertainty in certain focuses within prior analyses. All models in section 4 are evaluated for the long-term model, whilst only some pointwise and ranking models are chosen for the short-term model.

6.1 Long-Term Results

The long-term model results are obtained from the multi-week evaluation framework described in section 3.3.2. The below results tables highlight the important milestones for Gousto's business requirements, since menu weeks are decided 8 weeks in advance, and ingredients begin to be sourced and stored in inventory from this point. The 3 week mark is when the menu week goes live for customer selection, and where the short term model predictions begin, with the 1 week mark demonstrating the best possible performance of the long-term models, despite being overridden by the short-term model at this point. The full results for all week lags are graphically displayed in Appendix A.1.

Figure 6.1 illustrates the 1-8 week lagged MAE performance of all models, across 52 iterations of the target week:

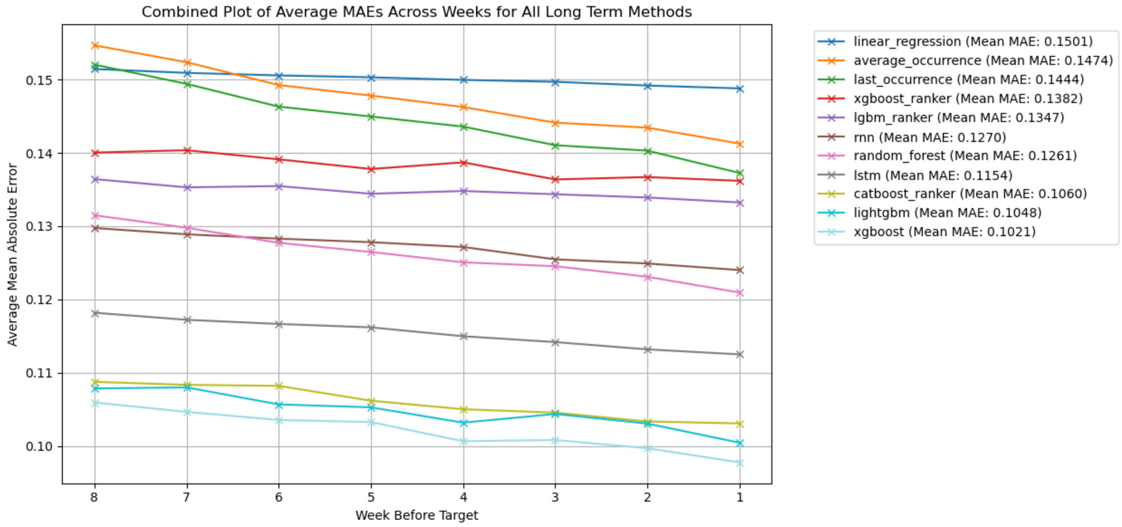

Figure 6.1: Mean Absolute Error (MAE) comparison of all models evaluated within the long-term multi-week pipeline across 52 iterations, with lags shown from 1 to 8 weeks before the target, alongside Mean MAE across all 52 iterations and 8 weeks within the legend. The Total Average model is omitted due to scale.

The following sections will explore each model's performance within its subgroup in more detail, however it is noted that the MAE values seem to mostly span the 0.10-0.15 range, which indicates that predicted ranks are typically within approximately $\lfloor \frac{N}{10} \rfloor$ ranks from the real rank, where N is the number of recipes in the menu week. For example with around 120 recipes per menu within recent weeks, the model expects to give rank predictions for recipes within 12 ranks from the true rank. This provides justification for choosing the MAE as the evaluation metric within section 3.3.1, due to its high interpretability within the given context.

6.1.1 Statistical Models Comparison

Table 6.1 displays the results for all three statistical models from section 4.1 at 8, 3 and 1 week lags:

Model	8 Week Lag	3 Week Lag	1 Week Lag
Total Average	0.2499	0.2499	0.2499
Last Occurrence	**0.1520**	**0.1410**	**0.1373**
Average Occurrence	0.1547	0.1441	0.1413

Table 6.1: Average Mean Absolute Error for Statistical Models at 8, 3 and 1 Week Lags

This identifies the Last Occurrence and Average Occurrence models as performing significantly better than the Total Average model, with the Last Occurrence model being marginally the best. The Total Average model performing the worst aligns with expectations since this was intended to be an absolute baseline, effectively predicting a rank percentile of 0.5 for each recipe.

The Last Occurrence model performs consistently better than the Average Occurrence model by an overall MAE difference of 0.003 across all 8 weeks. This may indicate that recent menu weeks are a better indicator of future recipe ranks, as opposed to considering old and new menus with equal weighting. However, it is important to note that if there are new recipes in each test menu week, then both models will revert to the total average prediction (averaging all training outputs) since the test recipe may have not appeared before. Analysing the `new_recipe` feature within the dataset, 25.59% of recipes per menu week have never appeared previously, with 15.51% of recipes within the testing dataset (most recent 52 menu weeks) being new recipes. This intuitively aligns with the analysis from figure 5.1, since the number of recipes has increased significantly from 10 to 120 over time; more recipes per menu week is indicative that new recipes are being created, resulting in these statistical methods performing suboptimally for the substantial proportion of new recipes. These statistical models have an over-dependence on historical recipe data from the exact same `item_id`, which does not apply effectively to Gousto's product offering, where new recipes are commonly introduced and approximately 15% of recipes in a given menu week are expected to be unseen before. This suggests that more sophisticated models which can generalise to any recipe based on their features (attributes, ingredients, base information) are likely to perform better on a dataset with continuously changing recipes and menus.

6.1.2 Pointwise Models Comparison

Table 6.2 displays the results for all four supervised learning models and two neural networks from sections 4.2 and 4.3 at 8, 3 and 1 week lags:

Model	8 Week Lag	3 Week Lag	1 Week Lag
Linear Regression	0.1514	0.1497	0.1488
LightGBM	0.1079	0.1044	0.1005
XGBoost	**0.1060**	**0.1009**	**0.0978**
Random Forest	0.1315	0.1245	0.1210
RNN	0.1298	0.1255	0.1240
LSTM	0.1182	0.1142	0.1125

Table 6.2: Average Mean Absolute Error for Pointwise (Supervised Learning and Neural Network) Models at 8, 3 and 1 Week Lags

The performance of LightGBM and XGBoost, especially the latter, demonstrates the strength of gradient-boosting methods for this particular task. XGBoost consistently outperforms the other models across all lags, and its ability to drop below 0.10 MAE at a 1-week lag highlights its effectiveness in short-term predictions. This result underscores XGBoost's capacity to capture complex patterns within the data while benefiting from its regularisation techniques, which likely prevent overfitting despite the relatively small lag period.

In contrast, Random Forest shows a steady but less pronounced improvement as the lag decreases, with an MAE of 0.1210 at the 1-week lag. This suggests that the combined method's ability to average decision trees does not fully exploit the underlying data patterns for shorter-term forecasts, particularly when compared to boosting methods like XGBoost and LightGBM [Friedman, 2001]. Nevertheless, its performance is noticeably better than Linear Regression, which sees minimal improvement from an 8-week to a 1-week lag. The limited performance in Linear

Regression highlights its inability to model non-linear relationships effectively [Chen and Guestrin, 2016], a limitation that aligns with earlier observations from section 3.2.

Among the neural networks, the LSTM model clearly surpasses the RNN in all lag intervals, reflecting the LSTM's capacity to manage longer dependencies and mitigate the vanishing gradient problem. Its ability to maintain a lower MAE, even at a 1-week lag, suggests that it is able to capture time-dependent dynamics more effectively than the simpler RNN architecture. The RNN, while still competitive with Random Forest, lags behind in comparison to both LSTM and the gradient boosting models. This result reinforces the importance of advanced recurrent architectures like LSTM in capturing temporal dependencies in sequential data [Hochreiter and Schmidhuber, 1997]. It is worth noting that neural networks, particularly LSTMs and RNNs, typically require larger datasets to perform optimally. While the current dataset contains 27,330 rows, which is sufficient for gradient-boosting models, neural networks tend to benefit more from greater data volume. With more rows, their performance would likely improve further.

In summary, XGBoost continues to demonstrate superior performance across all lags, particularly excelling in short-term forecasts. While LightGBM performs closely behind, the other models, particularly Linear Regression, demonstrate the limitations of more basic modelling techniques. The LSTM model's performance among the neural networks confirms its value for time-series forecasting, suggesting that models capable of retaining long-term information should be prioritised for future iterations of this analysis. Overall, these results further suggest that the dataset benefits from models that can exploit both non-linearity and temporal dynamics.

6.1.3 Ranking Models Comparison

Table 6.3 displays the results for all learning-to-rank models from section 4.4 at 8, 3 and 1 week lags:

Model	8 Week Lag	3 Week Lag	1 Week Lag
LightGBM Ranker	0.1364	0.1344	0.1332
XGBoost Ranker	0.1400	0.1364	0.1362
CatBoost Ranker	**0.1088**	**0.1046**	**0.1031**

Table 6.3: Average Mean Absolute Error for Ranking Models at 8, 3 and 1 Week Lags

These results show that the CatBoost Ranker consistently outperforms both the LightGBM and XGBoost Rankers across all lag periods. CatBoost Ranker achieves the best performance overall, with significantly lower MAE values at every lag. This suggests that CatBoost Ranker is particularly well-suited to the ranking task, likely benefiting from its ability to handle categorical features and its gradient boosting technique, which may allow it to generalise better across varying menu weeks.

The LightGBM Ranker performs slightly better than the XGBoost Ranker at every lag. Its histogram-based learning algorithm likely provides faster convergence and a more memory-efficient approach, which could explain its consistent performance advantage. Although the differences in MAE between the two models are relatively small, the LightGBM Ranker's performance shows robustness in ranking tasks, making it a reliable choice across the different lags. It is an important takeaway that LambdaMART, a cutting-edge model, does not seem to be the most optimal model in this study, demonstrating that this particular dataset has features which still require extra treatment and processing to improve predictions.

XGBoost Ranker, while slightly trailing behind both CatBoost and LightGBM, still demonstrates a competitive MAE across all time periods. However, its relatively higher MAE at each lag suggests that XGBoost may not handle the ranking task as effectively as the other models in this particular dataset. The absence of specialised techniques like CatBoost's categorical handling or LightGBM's histogram-based approach might be limiting its performance in this scenario.

In summary, the CatBoost Ranker exhibits superior performance across all lag periods, significantly outperforming both LightGBM and XGBoost Rankers. While LightGBM Ranker maintains a slight edge over XGBoost Ranker, both models fall far behind CatBoost in terms of MAE performance. The results highlight the importance of leveraging models that can handle the complexities of ranking tasks, with CatBoost standing out as the best performer in this analysis.

6.1.4 Long-Term Results Summary

The most notable takeaway from the long-term model analysis is that ranking models appear to perform overall worse than gradient boosting pointwise models, despite learning-to-rank models being specialised for ranking tasks and handling ordinal data. The best-performing pointwise models, LightGBM and XGBoost, are nearly matched by the CatBoost Ranker, whereas their ranking counterparts perform significantly worse and are instead comparable to the Last Occurrence model when evaluating one week before the target.

This result may be due to several factors. One possibility is that the increasing number of recipes per menu week, especially towards the later weeks where group sizes reach ∼120 recipes, makes ranking between them more challenging. As the number of items to rank grows, correctly ordering them becomes more complex, potentially reducing the effectiveness of the ranking models [Qin and Liu, 2013].

Additionally, the ranking models were optimised on NDCG, which prioritises the correct order of top-ranked items, while the final evaluation metric was MAE, a pointwise measure of absolute error. Since ranking models focus on the relative order of items, they may not be optimised to minimise MAE directly, leading to suboptimal performance when evaluated on this metric [Liu, 2009]. Rankers tend to excel with ranking-based metrics, while MAE measures absolute differences between predicted and true ranks, which may explain why they underperform compared to the pointwise models.

In summary, while the CatBoost Ranker shows promise by achieving competitive results, the overall performance of ranking models lags behind pointwise models. This suggests that, despite the advantages of learning-to-rank algorithms, gradient boosting models such as LightGBM and XGBoost are better suited for this task, particularly when evaluated using MAE in a dataset where the number of recipes and the introduction of new items regularly challenge the models' ranking capabilities.

6.2 Short-Term Results

The short-term model results, obtained from experiments using the multi-day evaluation framework in sections 3.5.2 and 5.5.2, showcase the performance of the four supervised models from section 4.2 and the three ranking models from section 4.4. The mean MAE values over 52 iterations are computed for each model, from lead day 21 to -6, where lead days 21 to 18 only involve long term features, and 17 onwards incrementally include more real-time features, as described in algorithm 3.

Table 6.4 below displays the mean MAE values for each model, at selected lead days, attempting to predict the final target at `lead_day_-4/-5/-6`. These indicate important milestones for Gousto's menu week journeys, since it represents the day just before the menu week goes live (`lead_day_18`), the day after the menu week goes live (`lead_day_17`), 1 week before the first delivery gets sent out (`lead_day_7`), the day when the first delivery selections are locked (`lead_day_3`), the day when the first delivery is sent out (`lead_day_0`), and the day when all delivery selections are locked (`lead_day_-4`), respectively. Figure 6.2 illustrates each model performance over all lead days.

`Model`	`Lead_day_18`	`Lead_day_17`	`Lead_day_7`	`Lead_day_3`	`Lead_day_0`	`Lead_day_-4`
Real-Time Prediction	-	0.0705	0.0331	0.0179	0.0053	0.0001
Linear Regression	0.1549	0.0583	0.0251	**0.0140**	**0.0049**	**0.0001**
LightGBM	0.1044	**0.0514**	**0.0249**	0.0141	0.0051	**0.0001**
XGBoost	**0.1009**	0.0539	0.0261	0.0151	0.0056	0.0002
Random Forest	0.1437	0.0637	0.0311	0.0167	0.0053	0.0002
LightGBM Ranker	0.1355	0.0711	0.0400	0.0310	0.0258	0.0255
XGBoost Ranker	0.1569	0.0935	0.0803	0.0696	0.0595	0.0570
CatBoost Ranker	0.1258	0.0553	0.0275	0.0167	0.0078	0.0038

Table 6.4: Average Mean Absolute Error at Selected Lead Days 18, 17, 7, 3, 0 and -4. The best-performing model per lead day is bolded.

These results indicate that a combination of XGBoost (from the long-term results), LightGBM and Linear Regression all provide the optimal predictive performance across the short-term time frame. The real-time prediction represents if the current ranked uptake from customer selections were used as the final prediction. This provides a crucial benchmark for each model, since this is the base predictive feature within each model, before each model uses all other features and its own optimisation algorithms. If any model were to under-perform relative to the real-time predictions, this would indicate a severe mishandling of the short-term features where their significance has not been sufficiently learned by the model.

This behaviour is evident for the LightGBM Ranker and XGBoost Ranker, where the MAE values consistently exceed the real-time predictions significantly. Even at `lead_day_-4` where the final targets are effectively included as a feature, these models seems to greatly underperform when compared to the other models. These models may not be suited to this dataset due to the similar reasons as described in section 6.1.4, where the large menu weeks and NDCG optimisation may prohibit the model from learning from feature importance and lean towards overfitting.

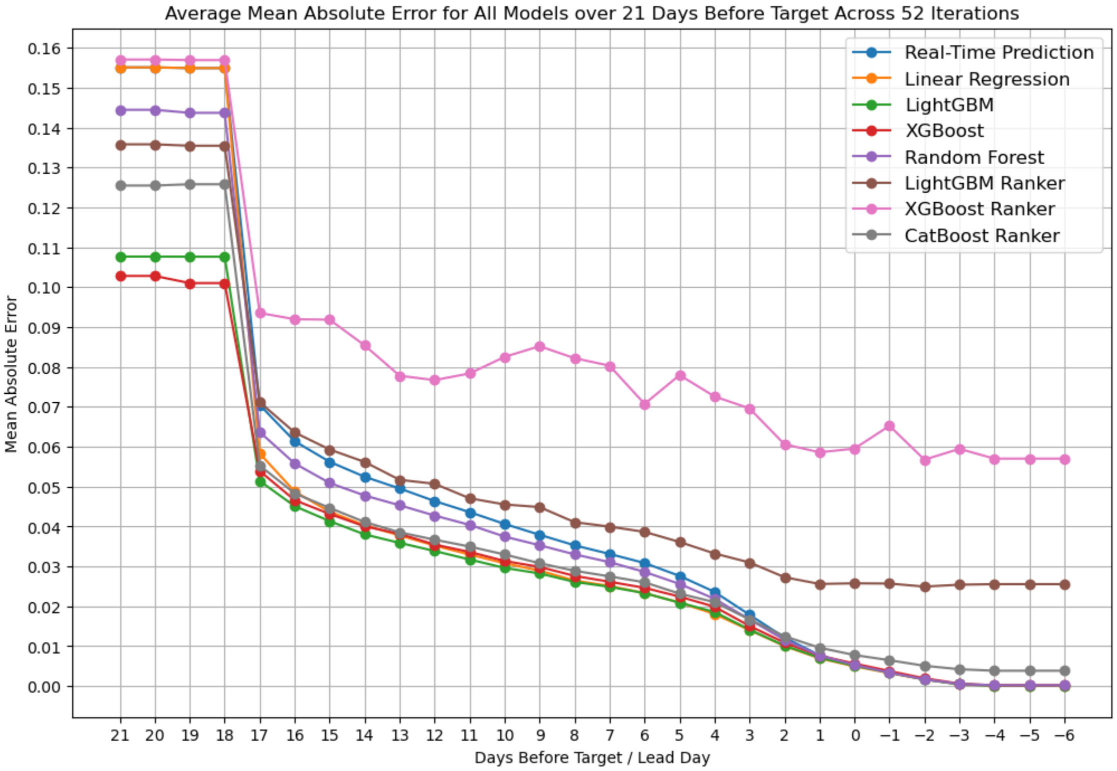

Figure 6.2: Average MAE for each model across all lead days over 52 iterations

XGBoost, Random Forest and CatBoost Ranker exhibit similar performances over each selected lead day, where model convergence at `lead_day_-4` is significantly better than the other ranking models, although CatBoost Ranker still has a noticeable discrepancy from the other pointwise models. Although these models outperform the real-time prediction benchmark for earlier lead days from 17 to 1, the real-time prediction becomes preferred after this point. This may indicate that these models are unable to significantly weight the real-time features with a shorter lag, and similarly the models may have too many features to prevent overfitting.

Linear Regression and LightGBM both appear to have effectively learned from the additional short-term features by outperforming the real-time prediction benchmark at all lead days. Both models experience a large 62% and 51% drop in MAE from `lead_day_18` to `lead_day_17` for Linear Regression and LightGBM respectively, which only continues to drop with significant improvement from the real-time prediction.

In order to evaluate when exactly LightGBM or Linear Regression outperform each other on which lead day, table 6.5 below displays the performance of Linear Regression and LightGBM:

Lead_day	Real-Time	Linear Reg	LightGBM
17	0.0705	0.0583	**0.0514**
16	0.0614	0.0488	**0.0451**
15	0.0562	0.0437	**0.0413**
14	0.0525	0.0403	**0.0380**
13	0.0495	0.0378	**0.0359**
12	0.0464	0.0352	**0.0339**
11	0.0436	0.0330	**0.0317**
10	0.0406	0.0308	**0.0296**
9	0.0379	0.0289	**0.0282**
8	0.0353	0.0265	**0.0261**
7	0.0331	0.0251	**0.0249**
6	0.0308	0.0233	**0.0233**
5	0.0276	0.0209	**0.0209**
4	0.0236	**0.0181**	0.0186
3	0.0179	**0.0140**	0.0141
2	0.0122	**0.0100**	0.0101
1	0.0077	**0.0070**	0.0071
0	0.0053	**0.0049**	0.0051
-1	0.0034	**0.0033**	0.0034
-2	0.0017	**0.0017**	0.0017
-3	0.0005	**0.0005**	0.0005
-4/-5/-6	0.0001	**0.0001**	**0.0001**

Table 6.5: Mean Absolute Error across the best-performing short-term models for each lead day from 17 to -6. The best model's results are bolded for each lead day.

The performance gap between LightGBM and Linear Regression becomes more pronounced as real-time features are incrementally introduced into the model predictions, starting from `lead_day_17`. LightGBM, with its gradient boosting approach and ability to capture complex feature interactions, excels in the early lead days where more long-term features are relied upon, achieving the lowest MAE values between `lead_day_17` and `lead_day_5`.

However, as more real-time data becomes available closer to `lead_day_0`, Linear Regression begins to show stronger performance. This can be attributed to its simplicity and ability to effectively handle linear relationships in the data, which might be more prominent in the short-term feature set as delivery selections get locked and customer preferences become clearer [Zhang and Qi, 2005]. The introduction of more short-term and real-time features reduces the complexity of the forecasting task, allowing Linear Regression to surpass LightGBM starting at `lead_day_4`.

Another possible reason for this shift is that LightGBM might be overfitting to the long-term features earlier in the process. As the model transitions into the short-term forecast, the additional real-time data could interfere with the relationships it has already learned, making it harder for the model to adapt. In contrast, Linear Regression, being less complex, adapts more efficiently to the new information, resulting in better predictions from `lead_day_4` onward.

Furthermore, the final convergence of both models at `lead_day_-4` highlights that once all delivery selections are locked, both models are essentially working with very similar datasets that contain limited variance in the input features. At this point, the predictive task becomes trivial with minimal room for error, potentially explaining why both models achieve the same 0.00011045 MAE at `lead_day_-4`, which is not zero due to data imperfections and calibration errors over the

52 test weeks.

LightGBM's ability to capture complex feature interactions gives it an edge in the earlier stages of the short-term forecast, but as the forecasting task becomes more linear and reliant on the short-term real-time data, Linear Regression takes over. This dynamic between model complexity and the availability of real-time features underscores the importance of selecting the appropriate model based on the specific time frame and feature set being utilised for prediction.

Unlike the long-term results where XGBoost outperformed other models at every weekly lag before target, the short-term model may require an combination of LightGBM and Linear Regression, due to their superiority with handling the real-time features.

Figure 6.3 displays the performance of the aforementioned models and the combined model, with the real-time prediction benchmark, focusing only from when the menu week goes live.

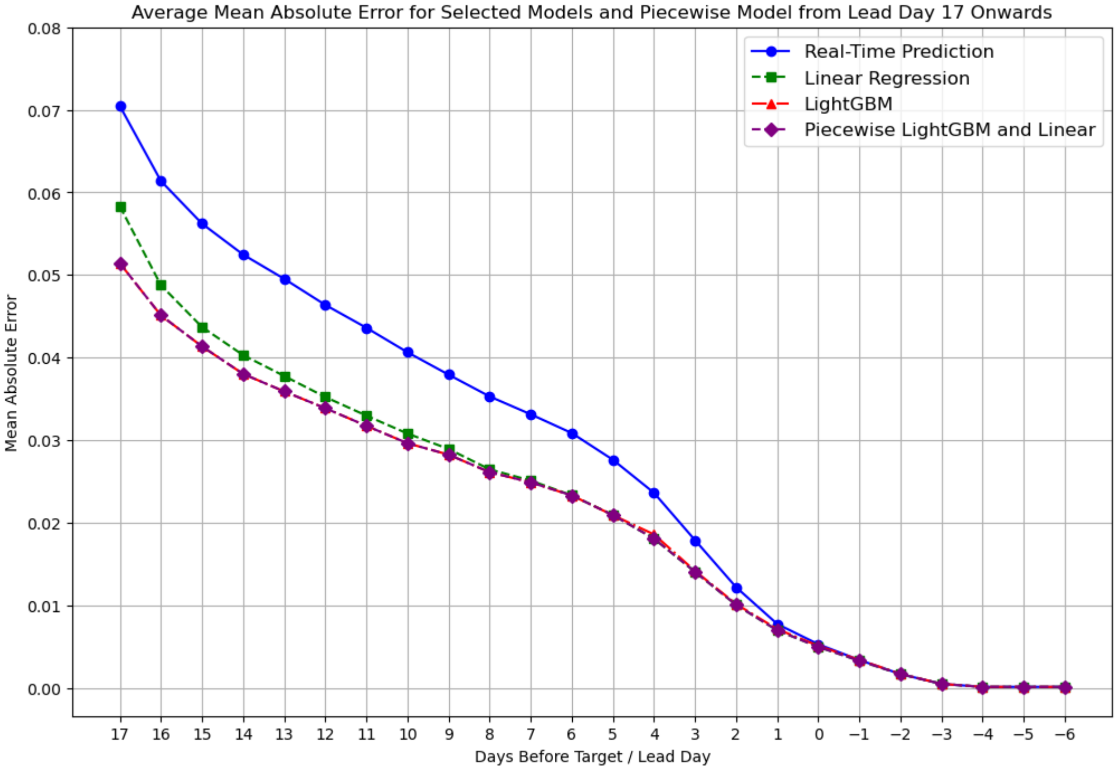

Figure 6.3: Average MAE for the best-performing models (Linear Regression and LightGBM) alongside the real-time prediction benchmark, and the combination of both models.

6.3 Hypothesis Test Results

For all hypothesis tests described in sections 3.6 and 5.6, this section presents the final results for the long-term and short-term models, concerning the 52 iterations of each experiment. These results include the models compared for each test alongside the week/day lag, with a 95% confidence interval for the MAE estimate, the associated T-statistic and p-value for the paired t-test.

6.3.1 Long-Term Hypothesis Test Results

Comparison of XGBoost and CatBoost Ranker

Table 6.6 shows the hypothesis test results for the top performing pointwise model and ranking model, for 8, 3 and 1 week lags. Section 6.1.2 concluded that XGBoost proved the best pointwise model, with CatBoost Ranker as the best ranking model, hence the hypothesis test is set as follows:

- Null Hypothesis (H_0): XGBoost performs equally as well as CatBoost Ranker.

$$H_0 : \text{MAE}_{\text{CatBoost Ranker}} = \text{MAE}_{\text{XGBoost}}$$

- Alternative Hypothesis (H_1): There is a significant disparity between the performance of XGBoost and CatBoost Ranker.

$$H_1 : \text{MAE}_{\text{CatBoost Ranker}} \neq \text{MAE}_{\text{XGBoost}}$$

The test is conducted at a 5% significance level (α) with 51 degrees of freedom (since there were 52 iterations), creating a critical region for this two-tailed test of [-2.0076, 2.0076] for the T-statistic. The sign of the T-statistic determines which model is stronger, in the case of a significant result. If t is below the critical region, XGBoost has the lower MAE, and vice versa if above.

Weeks Before	Model	Mean	Std Dev	95% CI	T-statistic	P-value
8	XGBoost CBRanker	0.1060 0.1088	0.0159 0.0167	(0.1015, 0.1104) (0.1041, 0.1134)	-2.7694	0.007813
3	XGBoost CBRanker	0.1009 0.1046	0.0129 0.0129	(0.0973, 0.1044) (0.1010, 0.1082)	-4.4455	0.000048
1	XGBoost CBRanker	0.0978 0.1031	0.0138 0.0164	(0.0940, 0.1016) (0.0986, 0.1077)	-5.3293	0.000002

Table 6.6: Hypothesis Test Results: XGBoost vs CatBoost Ranker

The results conclude that since the P-values are below the significance level of 0.05, that there is sufficient evidence to reject the null hypothesis H_0, suggesting that there is a disparity between the performance of both models. This applies for all selected week lags of 8, 3 and 1. Since the T-statistic is negative, XGBoost is deduced as the superior model with a noticeable yet small p-value of 0.007813, when predicting 8 weeks before the target week. However, at 1 week before the target, XGBoost's superiority over CatBoost Ranker is much greater with a p-value of 2.0×10^{-6}. Nonetheless, these hypothesis tests quantitatively conclude that XGBoost significantly outperforms CatBoost Ranker, at a 5% significance level.

Comparison of XGBoost and LightGBM Ranker (LambdaMART)

Table 6.7 shows the hypothesis test results for the top performing model and the LightGBM Ranker (LambdaMART) state-of-the-art model, for 8, 3 and 1 week lags. The prior section concluded XGBoost as the superior overall model for the long-term forecast, whilst knowing that LambdaMART did not even outperform CatBoost ranker. This hypothesis test still aims to quantify the disparity between the two models, as follows:

- Null Hypothesis (H_0): XGBoost performs equally as well as LightGBM Ranker.

$$H_0 : \text{MAE}_{\text{LightGBM Ranker}} = \text{MAE}_{\text{XGBoost}}$$

- Alternative Hypothesis (H_1): There is a significant disparity between the performance of XGBoost and LightGBM Ranker.

$$H_1 : \text{MAE}_{\text{LightGBM Ranker}} \neq \text{MAE}_{\text{XGBoost}}$$

The test is again conducted at a 5% significance level (α) with 51 degrees of freedom, creating a critical region for this two-tailed test of [-2.0076, 2.0076] for the T-statistic. The sign of the T-statistic determines which model is stronger, in the case of a significant result. If t is below the critical region, XGBoost has the lower MAE, and vice versa if above.

Weeks Before	Model	Mean	Std Dev	95% CI	T-statistic	P-value
8	XGBoost LGBMRanker	0.1060 0.1364	0.0159 0.0147	(0.1015, 0.1104) (0.1323, 0.1405)	-21.4744	0.000000
3	XGBoost LGBMRanker	0.1009 0.1344	0.0129 0.0154	(0.0973, 0.1044) (0.1301, 0.1386)	-24.3280	0.000000
1	XGBoost LGBMRanker	0.0978 0.1332	0.0138 0.0144	(0.0940, 0.1016) (0.1292, 0.1372)	-27.8157	0.000000

Table 6.7: Results Summary: XGBoost vs LightGBM Ranker

As expected, the T-statistics for these tests are very negative, indicating a huge improvement of XGBoost compared to LightGBM Ranker (LambdaMART). The p-value is negligible for all week lags, hence concluding a near zero probability of observing such extreme differences in MAE if the null hypothesis were true. This leads to confidently rejecting the null hypothesis H_0, confirming that XGBoost significantly outperforms LightGBM Ranker across all time lags.

6.3.2 Short-Term Hypothesis Test Results

Comparison of LightGBM and Linear Regression

Table 6.8 describes the short-term hypothesis test results for each `lead_day` from 17 to -6. Section 6.2 previously concluded that LightGBM and Linear Regression provided the best performances across different days before the first delivery date, hence necessitating this hypothesis test to better quantify the uncertainty of one model over the other. Simply by mean MAE comparisons, LightGBM proved superior until `lead_day_4` where Linear Regression outperformed LightGBM. However, this test aims to deduce at which days provide a statistically significant disparity between both models:

- Null Hypothesis (H_0): LightGBM performs equally as well as Linear Regression.

$$H_0 : \text{MAE}_{\text{Linear Regression}} = \text{MAE}_{\text{LightGBM}}$$

- Alternative Hypothesis (H_1): There is a significant disparity between the performance of

LightGBM and Linear Regression.

$$H_1 : \text{MAE}_{\text{Linear Regression}} \neq \text{MAE}_{\text{LightGBM}}$$

Similar to the previous tests, these were conducted at a 5% significance level with 51 degrees of freedom and a critical region of [-2.0076, 2.0076]. A T-statistic below the critical region suggested that LightGBM outperformed Linear Regression, and vice versa if above the critical region.

Lead_Day	Model	Mean	Std Dev	95% CI	T-statistic	P-value	Best Model
17	LGBM Linear	0.051423 0.058322	0.008581 0.009111	(0.049033, 0.053812) (0.055785, 0.060858)	-12.804500	0.000000	LightGBM
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
10	LGBM Linear	0.029621 0.030793	0.004812 0.004895	(0.028282, 0.030961) (0.029430, 0.032155)	-3.349800	0.001528	LightGBM
9	LGBM Linear	0.028233 0.028925	0.004023 0.004118	(0.027113, 0.029353) (0.027779, 0.030071)	-2.224200	0.030591	LightGBM
8	LGBM Linear	0.026114 0.026455	0.003968 0.003823	(0.025010, 0.027219) (0.025391, 0.027519)	-1.145000	0.257543	Similar (LGBM)
7	LGBM Linear	0.024878 0.025094	0.003336 0.003661	(0.023950, 0.025807) (0.024075, 0.026114)	-0.853500	0.397363	Similar (LGBM)
6	LGBM Linear	0.023295 0.023311	0.003342 0.003388	(0.022364, 0.024225) (0.022368, 0.024254)	-0.071500	0.943275	Similar (LGBM)
5	LGBM Linear	0.020911 0.020933	0.003278 0.003467	(0.019999, 0.021824) (0.019967, 0.021898)	-0.098200	0.922150	Similar (LGBM)
4	LGBM Linear	0.018578 0.018068	0.003412 0.003639	(0.017628, 0.019529) (0.017055, 0.019081)	2.485200	0.016269	Linear
3	LGBM Linear	0.014144 0.014040	0.002985 0.002989	(0.013313, 0.014976) (0.013200, 0.014880)	0.783000	0.437260	Similar (Linear)
2	LGBM Linear	0.010142 0.010038	0.002298 0.002316	(0.009502, 0.010781) (0.009393, 0.010683)	0.989500	0.327073	Similar (Linear)
1	LGBM Linear	0.007107 0.006956	0.001713 0.001795	(0.006630, 0.007584) (0.006456, 0.007456)	2.139100	0.037235	Linear
0	LGBM Linear	0.005084 0.004948	0.001296 0.001294	(0.004723, 0.005445) (0.004588, 0.005308)	2.819500	0.006832	Linear
-1	LGBM Linear	0.003378 0.003270	0.000891 0.000952	(0.003130, 0.003626) (0.003005, 0.003535)	3.182000	0.002491	Linear
-2	LGBM Linear	0.001685 0.001662	0.000696 0.000684	(0.001492, 0.001879) (0.001472, 0.001853)	2.075000	0.043050	Linear
-3	LGBM Linear	0.000467 0.000467	0.000262 0.000262	(0.000394, 0.000541) (0.000394, 0.000541)	-	-	Similar
-4/-5/-6	LGBM Linear	0.000110 0.000110	0.000116 0.000116	(0.000078, 0.000143) (0.000078, 0.000143)	-	-	Similar

Table 6.8: Results Summary: LightGBM vs Linear (Days Before 17 to -6, alpha = 0.05) - Shortened for highlighted results (full table in appendix A.2)

Here, Lead days 16 to 11 were omitted since these all suggested statistically significant results in favour of LightGBM, with negligible p-values, therefore providing evidence to reject H_0. Lead days 8 to 5 indicate no statistical significance between both models and hence H_0 is not rejected, with a marginal inclination towards LGBM, however this does suggest room for more iterations to have a more precise and certain conclusion, or that there may never be a meaningful difference between either model in this period.

Lead day 4 indicates a significant result in favour of Linear Regression, whereas lead days 3 and 2 revert to an insignificant result (still in favour of Linear Regression). This is surprising as it suggests that lead day 4 provides enough new information for linear regression to predict recipe rank much better than LightGBM, a huge swing from lead day 5, but only on lead day 3 does LightGBM catch back up to Linear Regression. Lead day 4 marks the first day where Gousto customers need to lock in their selections, hence may result in Linear Regression recognising this increase importance on the `uptake_at_lead_day` feature, better than LightGBM can.

Lead day 1 to -2 sees Linear Regression retake its place as being superior to LightGBM, regarding significance, hence the null hypothesis H_0 is rejected again in this period. From lead day -3 onwards, the model performances converge again to provide insufficient evidence to reject H_0, where the means are identical and hence providing a trivial hypothesis test. This null result is due to measurement error, where the precision of calculating error of ordinal rank percentiles shows its limit, and hence predictions become identical as the exact prediction precision is lost when ranked.

This analysis helps to quantify the exact uncertainty of model comparisons at each lead day, and clearly suggest numerous days (lead days 8 to 4, 3 to 2 and -3 to -6) where the difference is statistically insignificant. However, there are also many days where there is a clear 'winner', which also differs depending on the day (LightGBM for days 17 to 9, Linear on days 4 and 1 to -2). This provides further justification for a piecewise model across the short-term forecast, since either model can be statistically significant within this period.

Comparison of Piecewise Linear & LightGBM Model and Real-Time Data

Table 6.9 describes the short-term hypothesis test results for each `lead_day` from 17 to -6 for the piecewise model and the real-time uptake as direct predictions. Section 6.2 previously concluded that a piecewise model of LightGBM and Linear Regression provided the best performances across different days before the first delivery date, hence necessitating this hypothesis test to better quantify the uncertainty of this model's improvement over the customer's live selection data. The test is defined as follows:

- Null Hypothesis (H_0): The piecewise Linear & LightGBM model performs similarly to using the Real-Time Feature model.

$$H_0 : \text{MAE}_{\text{Piecewise}} = \text{MAE}_{\text{Real-Time}}$$

- There is a significant disparity between the performances of the Linear & LightGBM model and the Real-Time Feature model.

$$H_1 : \text{MAE}_{\text{Piecewise}} \neq \text{MAE}_{\text{Real-Time}}$$

Similar to the previous tests, these were conducted at a 5% significance level with 51 degrees of freedom and a critical region of [-2.0076, 2.0076]. A T-statistic below the critical region suggested that LightGBM outperformed Linear Regression, and vice versa if above the critical region.

Lead_Day	Model	Mean	Std Dev	95% CI	T-statistic	P-value	Best Model
17	Piecewise	0.051423	0.008581	(0.049033, 0.053812)	-28.6657	0.000000	Piecewise
	Real-Time	0.070469	0.009361	(0.067863, 0.073075)			
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	Piecewise	0.006956	0.001795	(0.006456, 0.007456)	-7.2354	0.000000	Piecewise
	Real-Time	0.007725	0.002072	(0.007148, 0.008302)			
0	Piecewise	0.004948	0.001294	(0.004588, 0.005308)	-4.8128	0.000014	Piecewise
	Real-Time	0.005275	0.001551	(0.004843, 0.005706)			
-1	Piecewise	0.003270	0.000952	(0.003005, 0.003535)	-3.0232	0.003907	Piecewise
	Real-Time	0.003400	0.001000	(0.003100, 0.003600)			
-2	Piecewise	0.001662	0.000684	(0.001472, 0.001853)	0.4113	0.682575	Similar (Real-Time)
	Real-Time	0.001661	0.000684	(0.001470, 0.001851)			
-3	Piecewise	0.000467	0.000262	(0.000394, 0.000541)	0.2479	0.805190	Similar (Real-Time)
	Real-Time	0.000467	0.000262	(0.000394, 0.000541)			
-4/-5/-6	Piecewise	0.000110	0.000116	(0.000078, 0.000143)	0.9934	0.325212	Similar (Real-Time)
	Real-Time	0.000110	0.000116	(0.000078, 0.000143)			

Table 6.9: Results Summary: Piecewise vs Real-Time Feature (Days Before 17 to -6, alpha = 0.05) - Shortened for highlighted results (full table in appendix A.2). Note that mean MAE values have been rounded in the table, but not in t-statistic calculation.

Evidently, the piecewise model provides a significant benefit to predicting the recipe rank percentile throughout the majority of the short-term forecast, from lead_day_17 to lead_day_-1. In this period, there is sufficient evidence to reject H_0, and since the T-statistic is negative, the piecewise model is concluded as superior.

However from lead_day_-2, the difference becomes insignficant and hence there is insufficient evidence to reject the null hypothesis H_0. In fact, there is a marginal inclination towards the Real-Time feature, which is remarkable since the piecewise model (which is linear in this range) directly uses the real-time data as a feature. Of course, MAE values at this magnitude are so low that any discrepancies are due to random modelling error and noise from other features, and alongside the fact that business impact is negligible, this can be safely ignored.

This set of hypothesis tests further reinforces the benefit of machine learning within this application, where the short-term forecast did significantly benefit from the implementation of the piecewise model for most days before the first delivery.

6.3.3 Hypothesis Test Results Summary

Overall, the four sets of hypothesis tests conducted have provided statistical justification to model selection and analysis, especially helping in differentiating between two similarly performing models and quantifying the impact of machine learning within this study.

Section 6.3.1 uncovered XGBoost as the best performing model in the long-term forecast, over both the best ranker model in CatBoost Ranker, and also against the state-of-the-art ranking model in LambdaMART via LightGBM Ranker. Section 6.3.2 deduced LightGBM and Linear Regression as the best models in different lead days within the short-term forecast. The hypothesis tests were able to discern between significant and insignificant overperformances between both models, and finally even indicated that the real-time features were (marginally) superior to the piecewise model, albeit with insufficient statistical evidence.

Chapter 7

Conclusion

This study examined the performance of various pointwise and ranking machine learning models in predicting recipe popularity rankings within Gousto's business model. The methodology and experimentation sections outlined key pre-processing steps and training algorithms, establishing robust evaluation frameworks suited to recipe ranking.

For the long-term model, pointwise gradient-boosting methods like XGBoost and LightGBM delivered the best performance, achieving a mean absolute error (MAE) of under 10% relative to true ranks. This result aligns with our initial analysis, which suggested that linear models would struggle to capture the non-linear relationships in the data, necessitating more advanced methods like gradient boosting.

In the short-term model, linear models proved most effective at handling real-time data, achieving the highest performance as predicted by prior correlation analysis. LightGBM also performed well in the early stages of short-term forecasting, capturing patterns from real-time data that XGBoost could not. This highlights LightGBM as the strongest single option for both long and short-term predictions.

A piecewise method combining XGBoost, LightGBM and Linear Regression would provide the optimal model performance across all stages of Gousto's menu releases. XGBoost handles predictions from 8 weeks from the first delivery date, up to the inclusion of real-time day at 17 days out. At `lead_day_4`, the piecewise model switches to Linear Regression until the final delivery date, although the hypothesis tests demonstrate the varying uncertainty of these conclusions. While this threshold may shift with additional data, the performance gap between models is small enough to suggest that future adjustments would have minimal impact.

A key takeaway is the underperformance of learning-to-rank models relative to pointwise models, despite being designed for ranking tasks. For instance, the long-term forecast performed better with a pointwise LightGBM model than with a state-of-the-art LambdaMART variant of LightGBM Ranker. This may be due to ranking models optimising for the NDCG objective rather than MAE, which leads to different weighting across recipes. Although NDCG improves interpretability by prioritising higher-ranked recipes, it does not focus on the overall accuracy required for minimising errors across all recipes, which may be more beneficial for controlling costs per recipe selection.

Furthermore, the large number of recipes per menu week may pose a challenge for pairwise ranking methods. With over 100 recipes in some weeks, many menus lack sufficient historical pairwise comparison data. This makes it difficult for ranking models like LambdaMART to effectively learn the ranking, particularly for unseen recipes. In contrast, pointwise models like LightGBM are more robust, as they directly predict the rank percentile without relying on pairwise comparisons, making them better suited for handling large menu weeks with frequent recipe turnover.

CatBoost Ranker stands out for its relative success among other ranking models, likely due to its ability to natively handle categorical features. Many core features in the dataset are categorical, and CatBoost's native handling of these relationships may give it an advantage over other ranking models that require manual encoding. Nonetheless, there is still a noticeable performance gap between CatBoost Ranker and the best performing pointwise models of XGBoost, LightGBM and Linear Regression.

Chapter 8

Future Work

This study has demonstrated the effectiveness of several machine learning models for predicting recipe rankings and evaluation frameworks, however there are several avenues for future work which could extend the results, analysis and conclusions drawn.

8.1 Rank Predictions Meta-Feature

One key aim of this project was to implement the predictions from the highest performing machine learning models as a meta-feature within Gousto's primary model, to improve continuous uptake predictions. Gousto makes the large majority of their cost-savings with more accurate predictions close to the first delivery date, since these involve predicting ingredients with low shelf-lives. For this analysis, the short-term model performances are compared before and after adding these rank prediction features.

Using the LightGBM predictions from the short-term model, Gousto's primary model exhibited varying degrees of improvement from the control (original) model, as observed in figure 8.1 below.

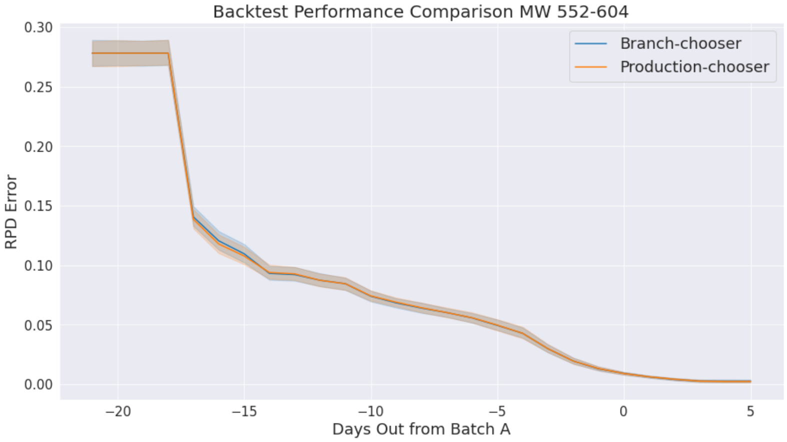

Figure 8.1: Comparison of short-term model performance of Gousto's original model between control model (Production-chooser) and model with rank prediction features (Branch-chooser), over 26 iterations

The new model does not display any significant difference from the old model; in fact it performs relatively worse in some sections of the plot, for example from lead days 17 to 15. This can be attributed to rank predictions losing detail when converting uptakes to ranks and subsequently rank percentiles. Models may be placing high importance on the new rank features, however when any prediction is wrong, then they will always be wrong by a substantial fixed amount since these are effectively ordinal/discrete values.

Improvements to these predictions are suggested in subsequent sections of this chapter, however one primary focus of this study involved optimisation of predicting ranks, which is a separate challenge to expecting ranks to be an influential feature for recipe uptake via backtesting. The study was limited by the unavailability of access to Gousto's internal model, to optimise on overall uptake performance, as opposed to purely rank performance.

8.2 Improvements to Ranking Models

Chapters 6 and 7 conclude that pointwise gradient-boosting models performed the best for predicting rank percentiles, however a deeper exploration into why ranking models performed poorly in relation to pointwise models may better inform a new approach to optimising ranking models.

In theory, ranking models are designed to handle ordinal data and better account for interdependencies between recipes within the same menu week, and hence provide more accurate rank predictions, however this study suggests that the NDCG evaluation metric and the large number of recipes per menu week rendered ranking models less effective for this particular dataset, as described in section 6.1.4.

Since Gousto's internal metric is based on MAE (WMAPE) and not NDCG, the performance of NDCG-optimised ranking models may suffer and suggest that ranking models may not work at all. This motivates a custom ranking model which does optimise on MAE instead, requiring further engineering to build a new model for this purpose, since the existing models do not cater for all possible evaluation metrics.

Despite the pointwise models outperforming the ranking models, they still provide pointwise predictions and hence do not factor in the predictions of other recipes in the menu week. Therefore, when Gousto's internal uptake model use these predictions, it still does not capture the interactions with other potential recipes in the menu week, whereas the ranking models may do so. If the rank prediction features could act as strict boundaries within Gousto's internal model, to constrain the final uptake predictions within each recipe's ranked prediction, this may allow the rank features to influence the uptake model at a greater level. However, again due to access limitations of this study with Gousto's internal model, this remains an area to be explored.

8.3 Feature Selection and Engineering

Regarding all features used in the long-term and short-term models, 202 features are related to recipe and attribute embeddings, which is a significant proportion of available features. An alternative approach to feature generation includes providing binary or categorical variables which may describe each recipe, for example being vegan or gluten-free, or the base of the dish (rice, pasta, bread etc). With CatBoost Ranker outperforming other rankers in the long-term model, this may suggest that the presence of more categorical features may benefit the model more than vector embeddings, since the latter relies on a well-defined latent space of recipes.

The inclusion of more features should also be explored, for example the estimated time to cook, or the calories per portion quantity, or the cooking mode (microwave, one-pot etc). These features are available on the Gousto website, therefore are not only easy to retrieve and load into the dataset, but also are signposted very clearly on each recipe and are likely to make a psychological difference in a customer's decision making process when selecting recipes. Ultimately, machine learning is attempting to mimic human behaviour, therefore features which appear right in front of the customer should play a huge role in recipe selection. This study explored the limitations of the provided dataset which Gousto use for their primary model, however extensions to both models could be explored with extra features.

8.4 Final Remarks

This study has demonstrated the most effective machine learning models for predicting Gousto recipe ranks, within long-term and short-term forecasts. The statistically significant underperformance of learning-to-rank algorithms within a ranking task is remarkable, however future work could help to further optimise the ranking models to leverage their intended benefits. Using rank predictions as a feature may help to create a significant improvement to Gousto's uptake predictions, however without improved implementation of ranking models and access to Gousto's internal model, this remains an area for future exploration. Overall, this study has achieved its aims in forecasting recipe rank within Gousto's business model.

Bibliography

Eric Michael Smith, Alfredo Nantes, Andrew Hogue, and Ilia Papas. Forecasting customer behaviour in constrained e-commerce platforms. In *8th International Conference of Pattern Recognition Systems (ICPRS 2017)*, pages 1–8, 2017. doi: 10.1049/cp.2017.0163.

Mehmet Aci and Derya Yergök. Demand forecasting for food production using machine learning algorithms: A case study of university refectory. *Tehnički vjesnik*, 30(6):1683–1691, 2023. doi: 10.17559/TV-20230117000232. URL https://hrcak.srce.hr/309217.

Sushil Punia and Sonali Shankar. Predictive analytics for demand forecasting: A deep learning-based decision support system. *Knowledge-Based Systems*, 258:109956, 2022. ISSN 0950-7051. doi: https://doi.org/10.1016/j.knosys.2022.109956. URL https://www.sciencedirect.com/science/article/pii/S0950705122010498.

Vansh Nagpal, Siva Likitha Valluru, Kausik Lakkaraju, and Biplav Srivastava. Beacon: Balancing convenience and nutrition in meals with long-term group recommendations and reasoning on multimodal recipes, 2024. URL https://arxiv.org/abs/2406.13714.

Sandeep Kumar Panda and Sachi Nandan Mohanty. Time series forecasting and modeling of food demand supply chain based on regressors analysis. *IEEE Access*, 11:42679–42700, 2023. doi: 10.1109/ACCESS.2023.3266275.

M Adith, Panduranga Katti, and Deepa Gupta. A strategic analysis of food demand using machine learning and explainable ai. In *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, pages 1–7, 2024. doi: 10.1109/I2CT61223.2024.10543689.

R. Denis and D. Keerthana. Performance analysis of voting regression-based ensemble learning methods for food demand forecasting. *International Journal of Data Informatics and Intelligent Computing*, 3(2):34–41, Jun. 2024. doi: 10.59461/ijdiic.v3i2.114. URL https://ijdiic.com/index.php/research/article/view/114.

Chris J. C. Burges. From ranknet to lambdarank to lambdamart: An overview. In *Learning to Rank Workshop (NIPS 2010)*, 2010.

Yijun Tian, Chuxu Zhang, Zhichun Guo, Yihong Ma, Ronald Metoyer, and Nitesh V. Chawla. Recipe2vec: Multi-modal recipe representation learning with graph neural networks, 2022. URL https://arxiv.org/abs/2205.12396.

Cort J. Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate Research*, 30 (1):79–82, 2005. doi: 10.3354/cr030079.

Bradley Efron and Robert J. Tibshirani. *An Introduction to the Bootstrap*. CRC Press, New York, 1994.

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631. ACM, 2019. doi: 10.1145/3292500.3330701.

James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554, 2011. URL https://proceedings.neurips.cc/paper/2011/file/86e8f7ab32cfd12577bc2619bc635690-Paper.pdf.

Christoph Bergmeir, Rob J. Hyndman, and Bonsoo Koo. A note on the validity of cross-validation for evaluating autoregressive time series prediction. *Computational Statistics & Data Analysis*, 120:70–83, 2018.

Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, Inc., 2018.

Spyros Makridakis, Steven C. Wheelwright, and Rob J. Hyndman. *Forecasting: Methods and Applications*. Wiley, 3rd edition, 1998.

Fotios Petropoulos and Nikolaos Kourentzes. Forecast combinations for intermittent demand. *Journal of the Operational Research Society*, 66(6):914–924, 2015.

Rob J. Hyndman and George Athanasopoulos. *Forecasting: Principles and Practice*. OTexts, 2nd edition, 2018. URL https://otexts.com/fpp3/.

Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning with Applications in R*. Springer, 2013.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems*, pages 3146–3154, 2017.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794, 2016.

Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29(5):1189–1232, 2001. doi: 10.1214/aos/1013203451.

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

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179–211, 1990.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 2012.

Klaus Greff, Rupesh K Srivastava, Jakub Koutnik, Bernd R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE Transactions on Neural Networks and Learning Systems*, 28(10):2222–2232, 2017.

Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2009.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *arXiv preprint arXiv:1706.09516*, 2018.

LightGBM Developers. *LightGBM LambdaRank Parameter Documentation*, 2024. URL https://lightgbm.readthedocs.io/en/latest/Parameters.html#lambdarank. Accessed: 2024-08-15.

Nithish Kannen, Yao Ma, Gerrit J. J. van den Burg, and Jean Baptiste Faddoul. Efficient pointwise-pairwise learning-to-rank for news recommendation, 2024. URL https://arxiv.org/abs/2409.17711.

Christoph Bergmeir and José M. Benítez. On the use of cross-validation for time series predictor evaluation. *Information Sciences*, 191:192–213, 2012.

Jason Brownlee. *Introduction to Time Series Forecasting with Python*. Machine Learning Mastery, 2017.

Tao Qin and Tie-Yan Liu. Introducing letor 4.0 datasets and performance metrics. *Information Retrieval Journal*, 17(4):359–384, 2013.

Gang Zhang and Min Qi. Neural network forecasting for seasonal and trend time series. *European Journal of Operational Research*, 160(2):501–514, 2005.

Appendix A

Appendix

A.1 Long-Term Multi-Week Forecast Plots

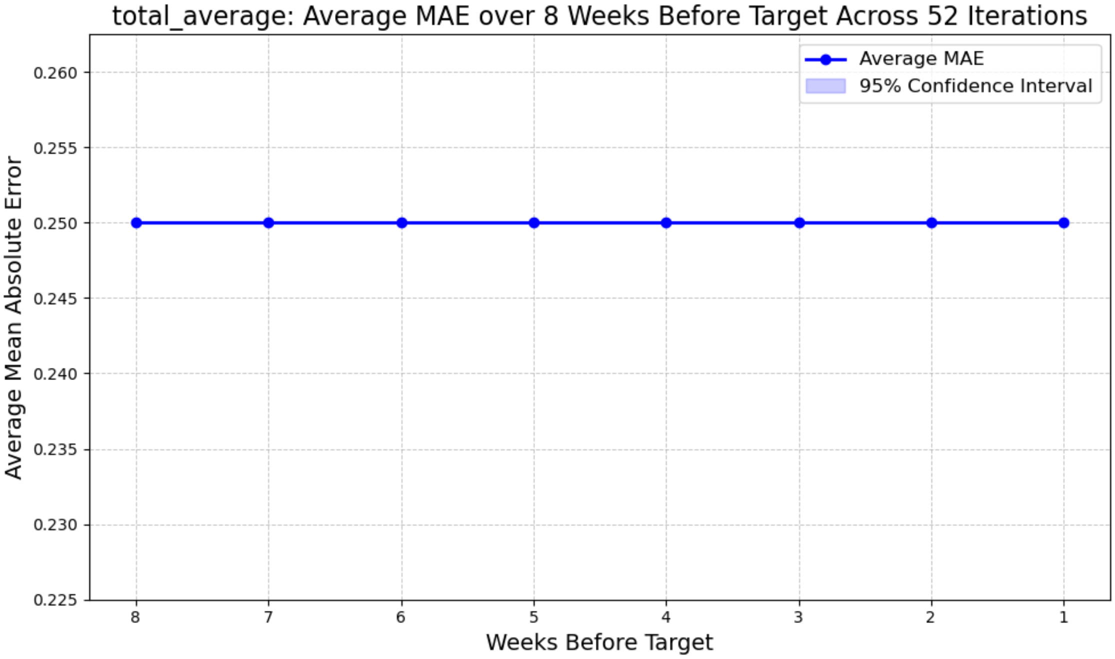

Figure A.1: MAE of totalaverage from 1 to 8 weeks before target, across 52 iterations

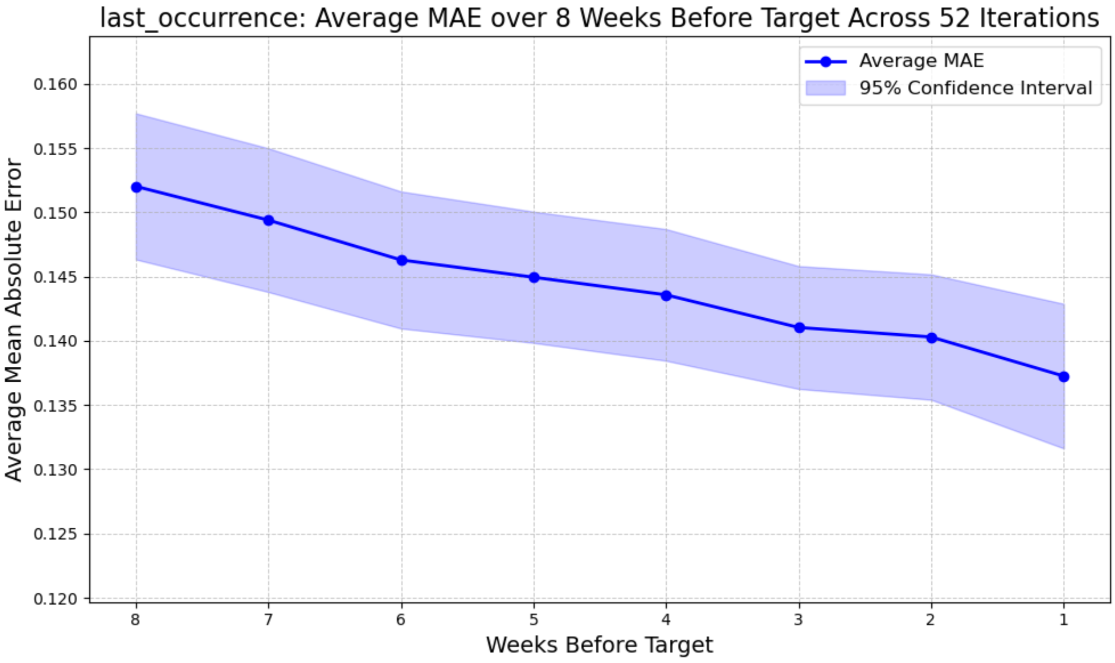

Figure A.2: MAE of lastoccurrence from 1 to 8 weeks before target, across 52 iterations

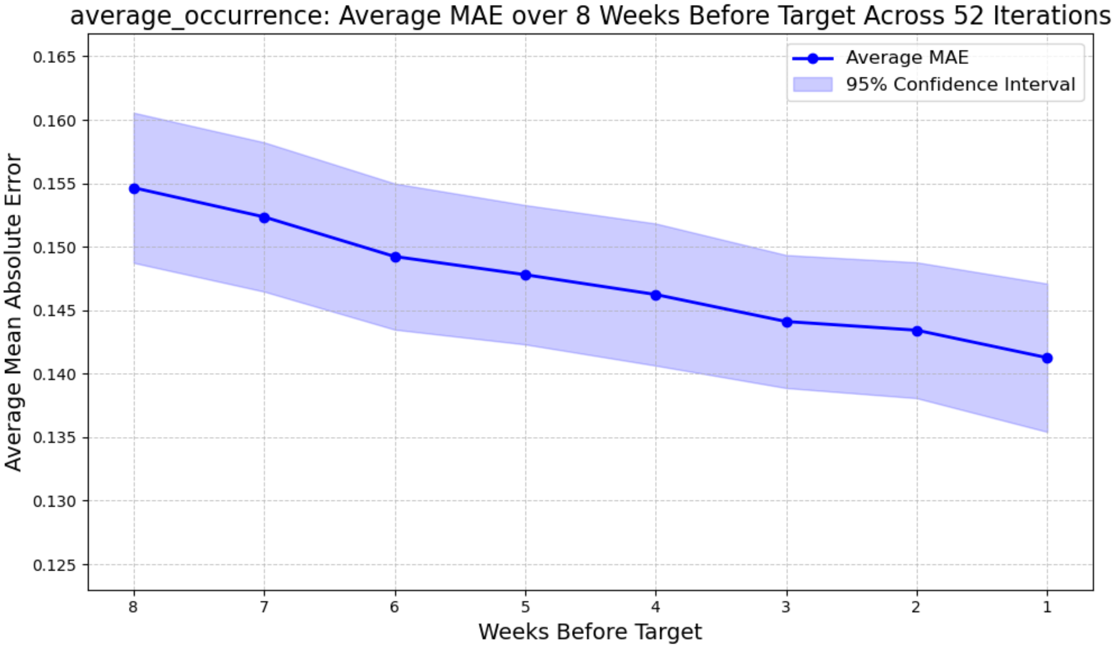

Figure A.3: MAE of avgoccurence from 1 to 8 weeks before target, across 52 iterations

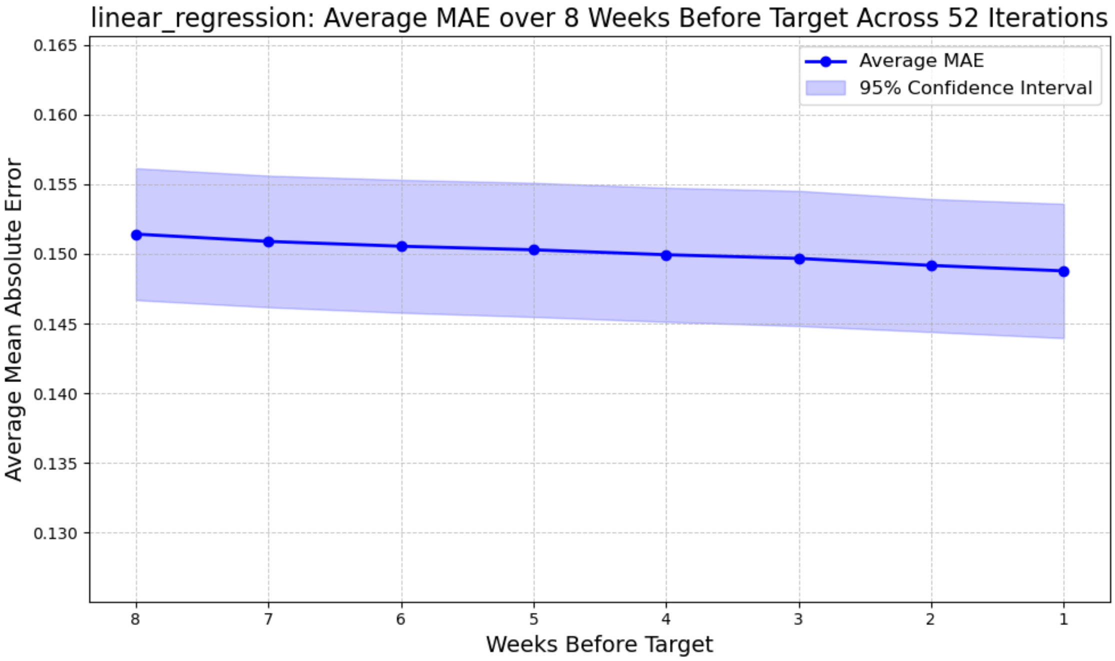

Figure A.4: MAE of linearregression from 1 to 8 weeks before target, across 52 iterations

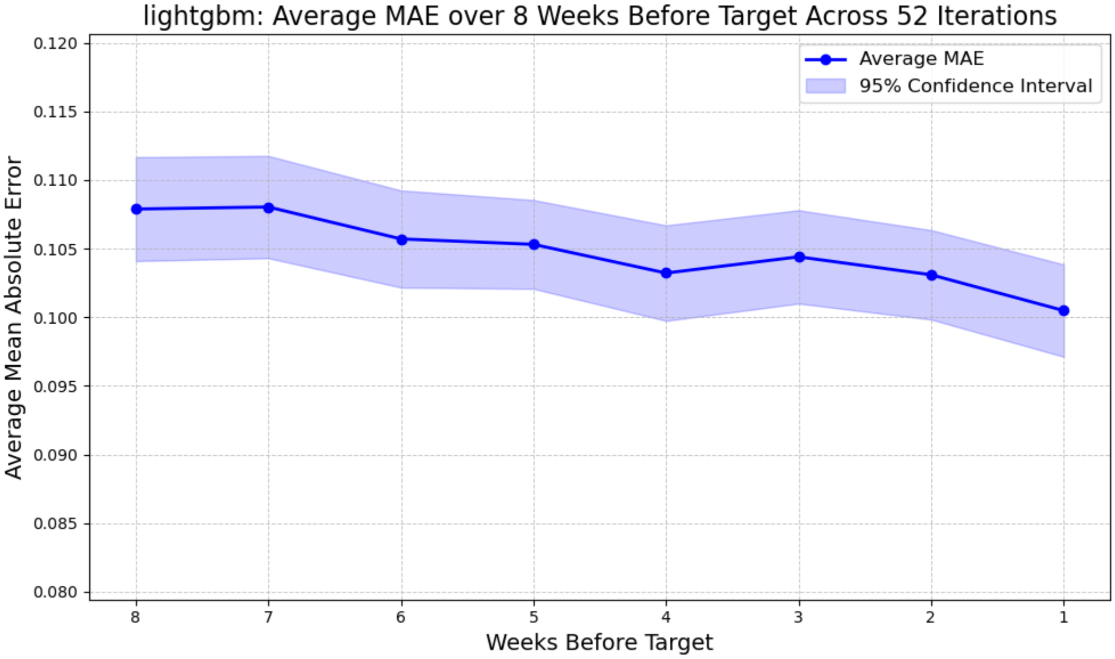

Figure A.5: MAE of lightgbm from 1 to 8 weeks before target, across 52 iterations

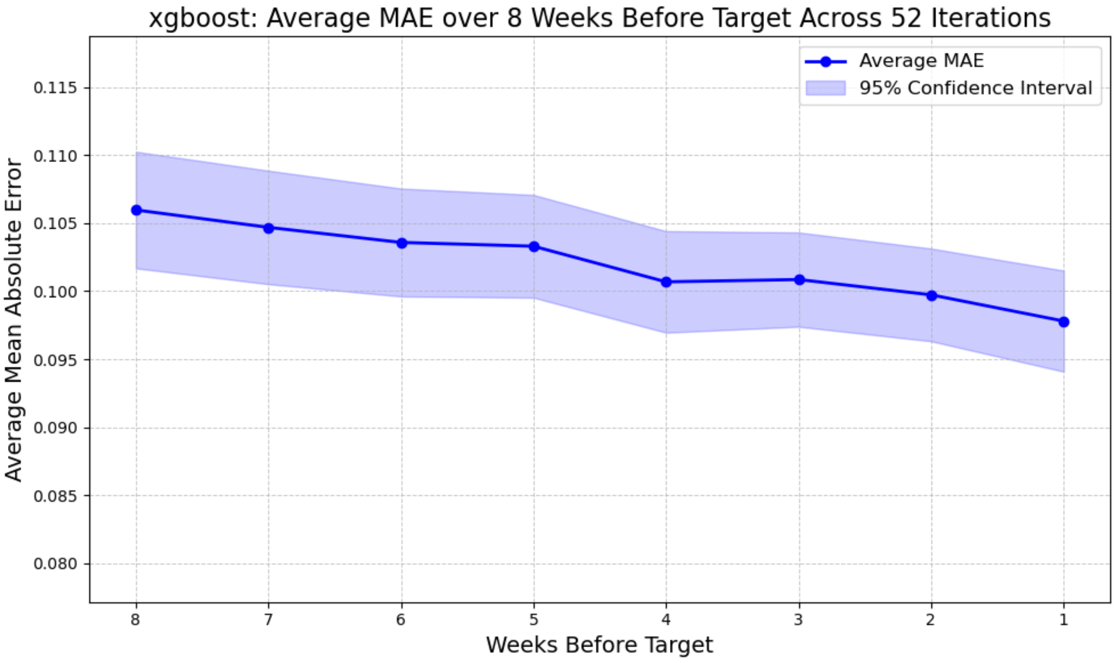

Figure A.6: MAE of xgboost from 1 to 8 weeks before target, across 52 iterations

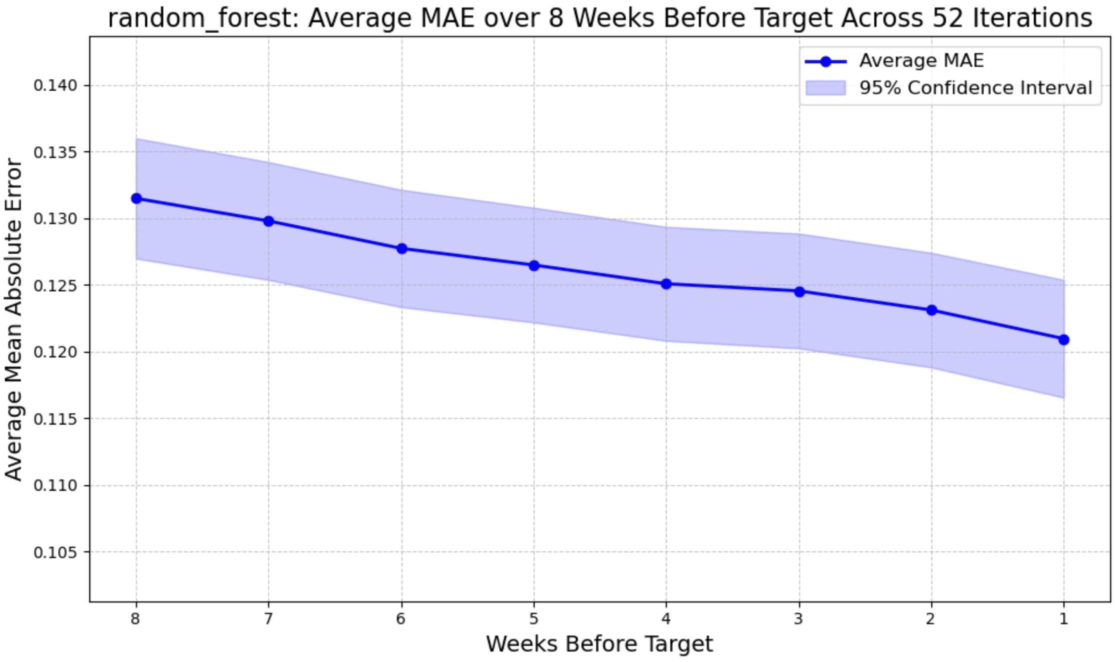

Figure A.7: MAE of randomforest from 1 to 8 weeks before target, across 52 iterations

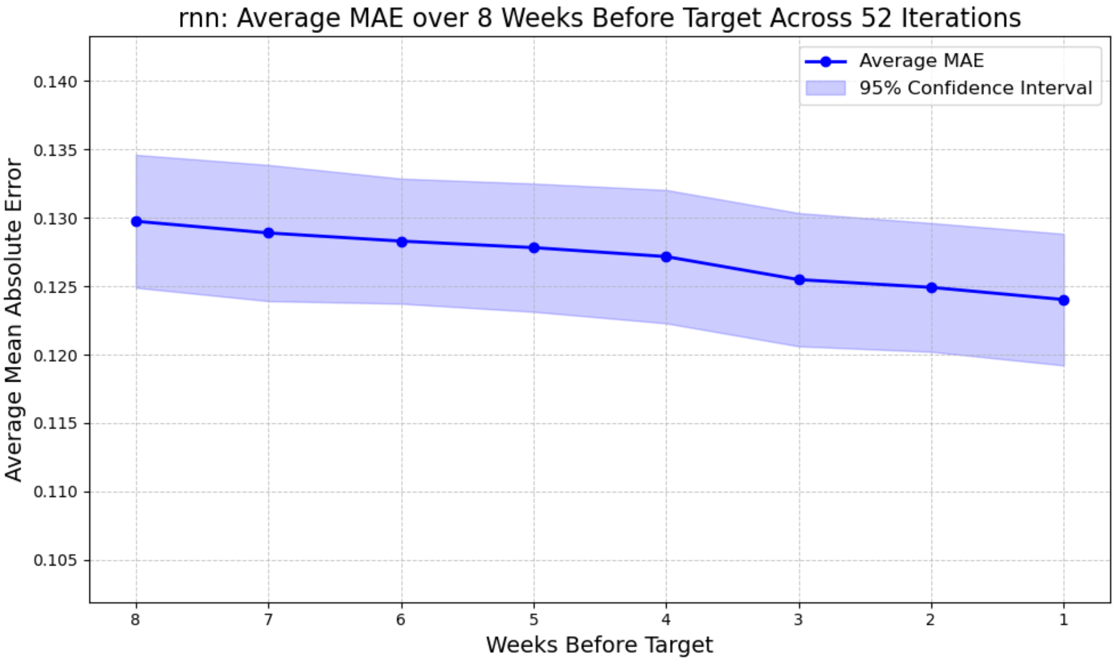

Figure A.8: MAE of rnn from 1 to 8 weeks before target, across 52 iterations

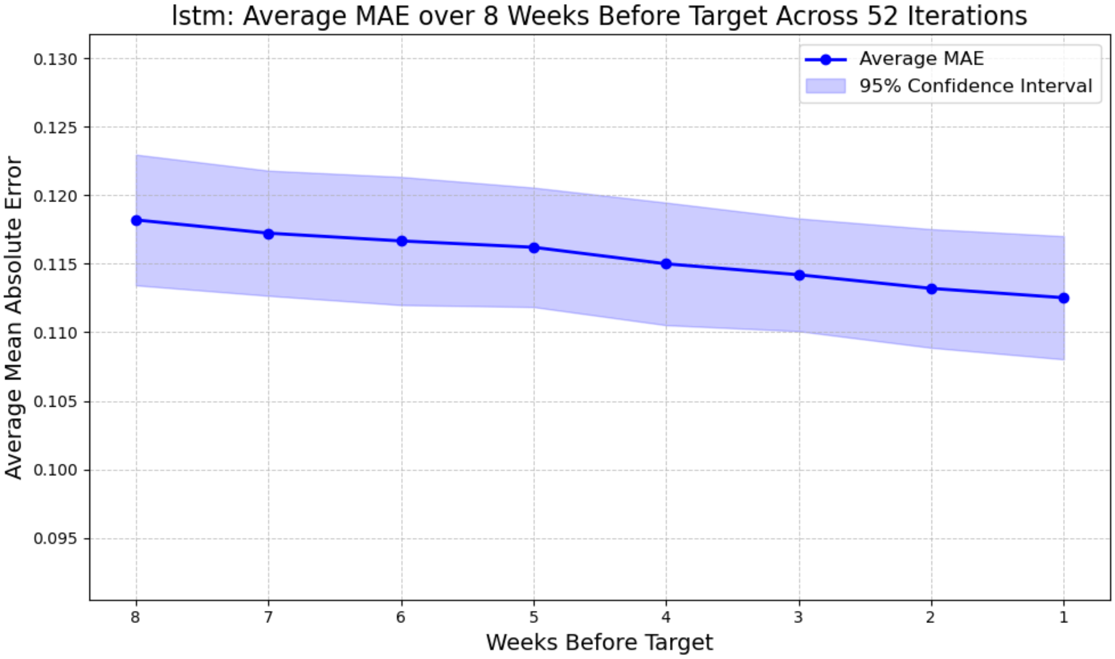

Figure A.9: MAE of lstm from 1 to 8 weeks before target, across 52 iterations

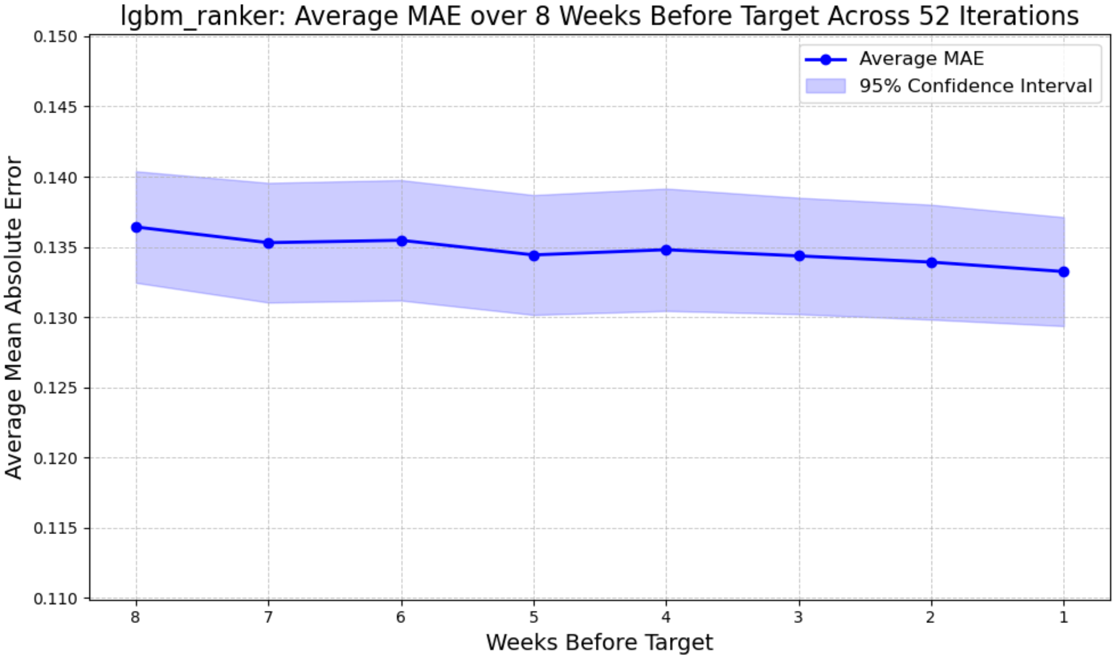

Figure A.10: MAE of lgbmranker from 1 to 8 weeks before target, across 52 iterations

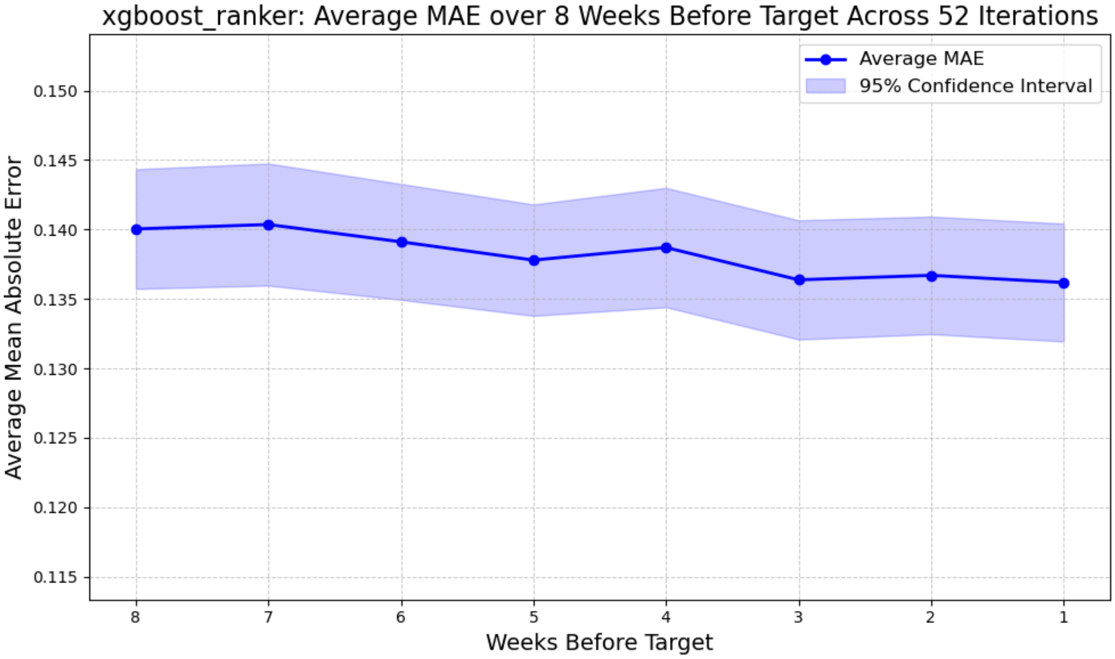

Figure A.11: MAE of xgboostranker from 1 to 8 weeks before target, across 52 iterations

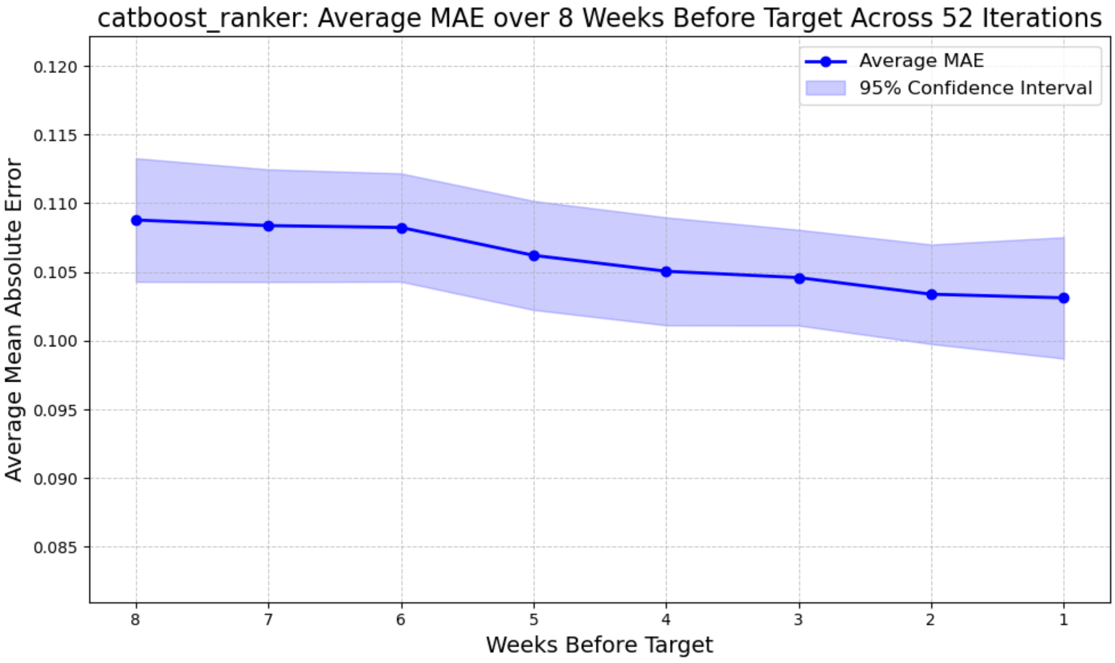

Figure A.12: MAE of catboostranker from 1 to 8 weeks before target, across 52 iterations

A.2 Short-Term Multi-Day Forecast Plots

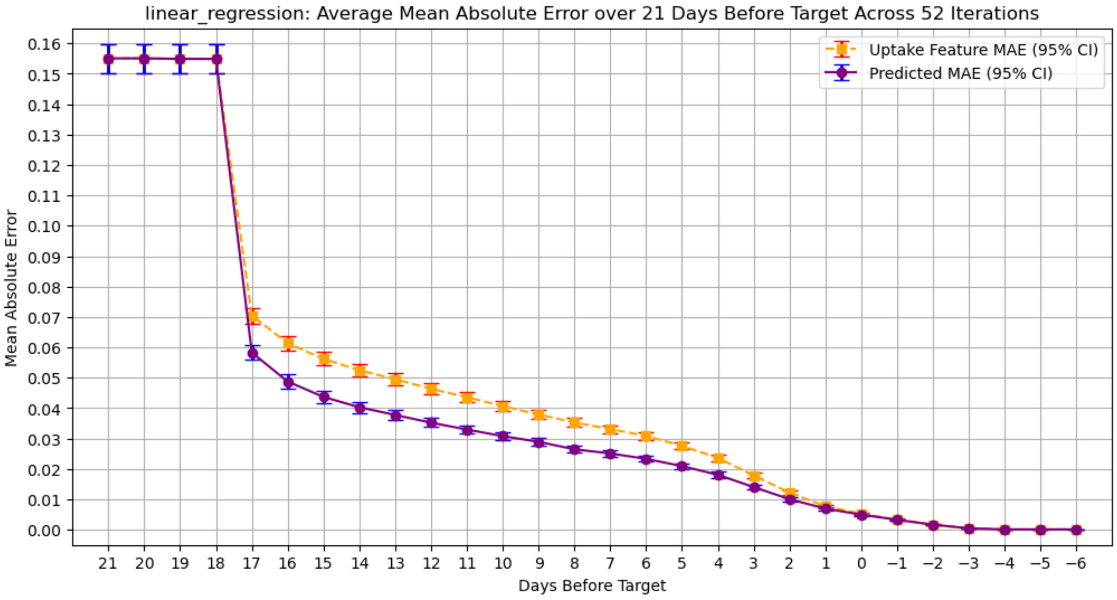

Figure A.13: MAE of linear52live from 1 to 8 weeks before target, across 52 iterations

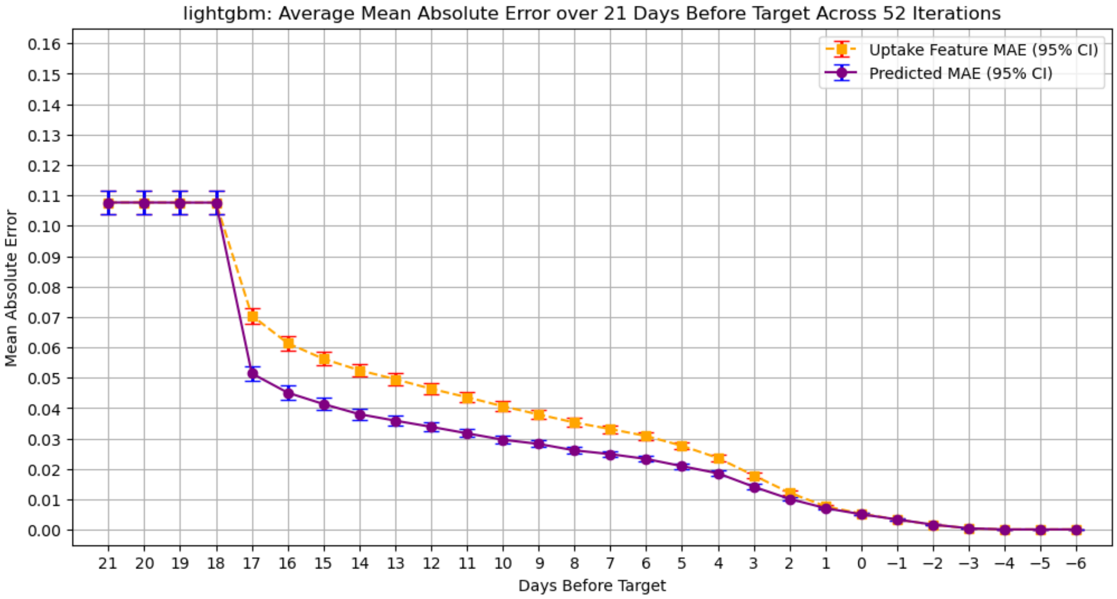

Figure A.14: MAE of lgbm52live from 1 to 8 weeks before target, across 52 iterations

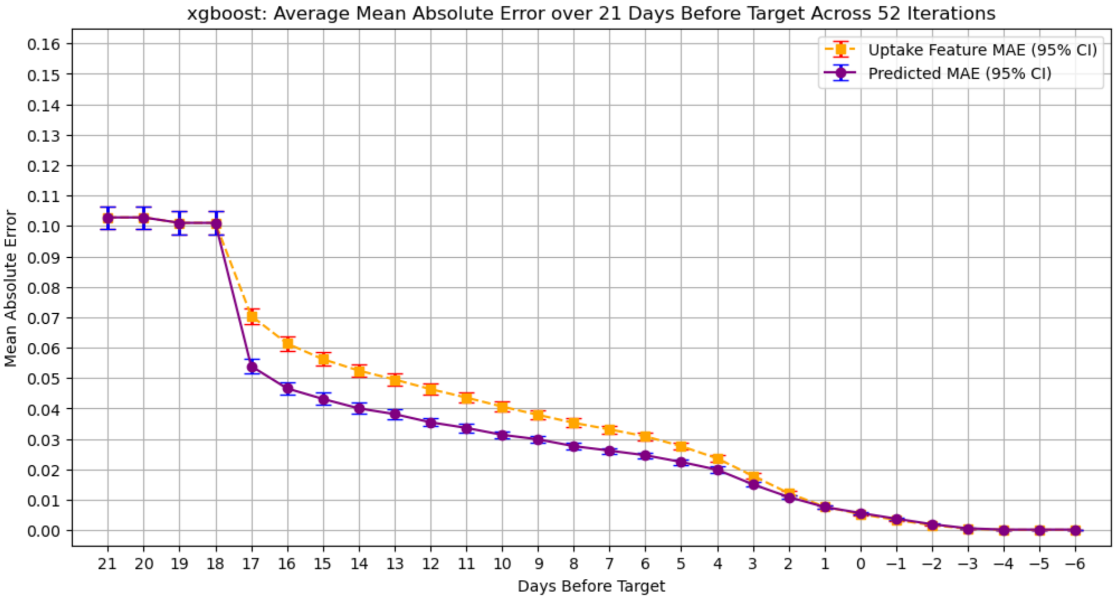

Figure A.15: MAE of xgboost52live from 1 to 8 weeks before target, across 52 iterations

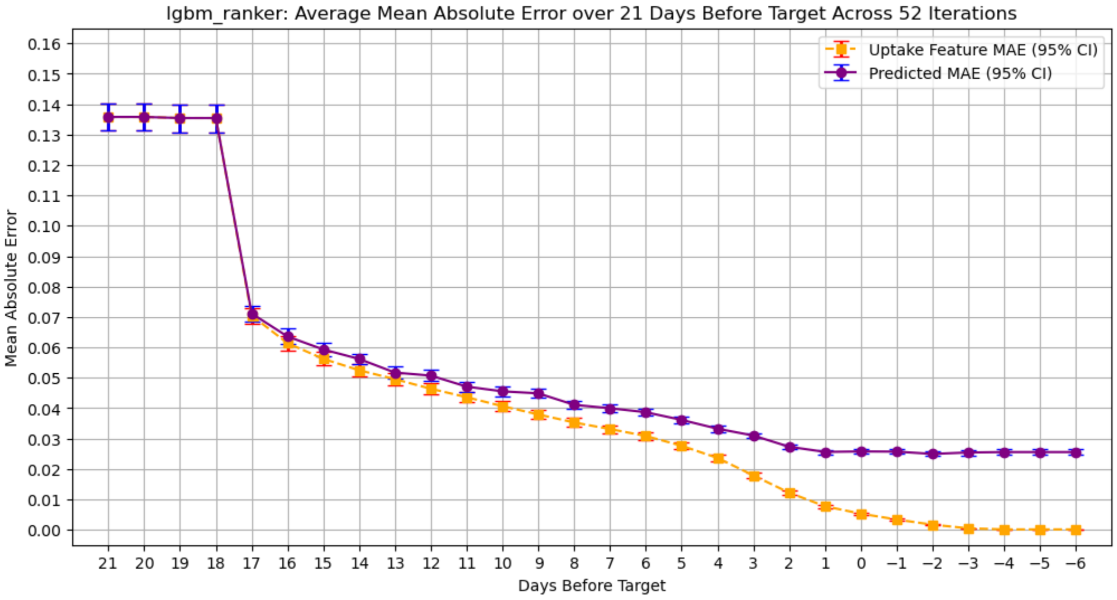

Figure A.16: MAE of lgbmranker52live from 1 to 8 weeks before target, across 52 iterations

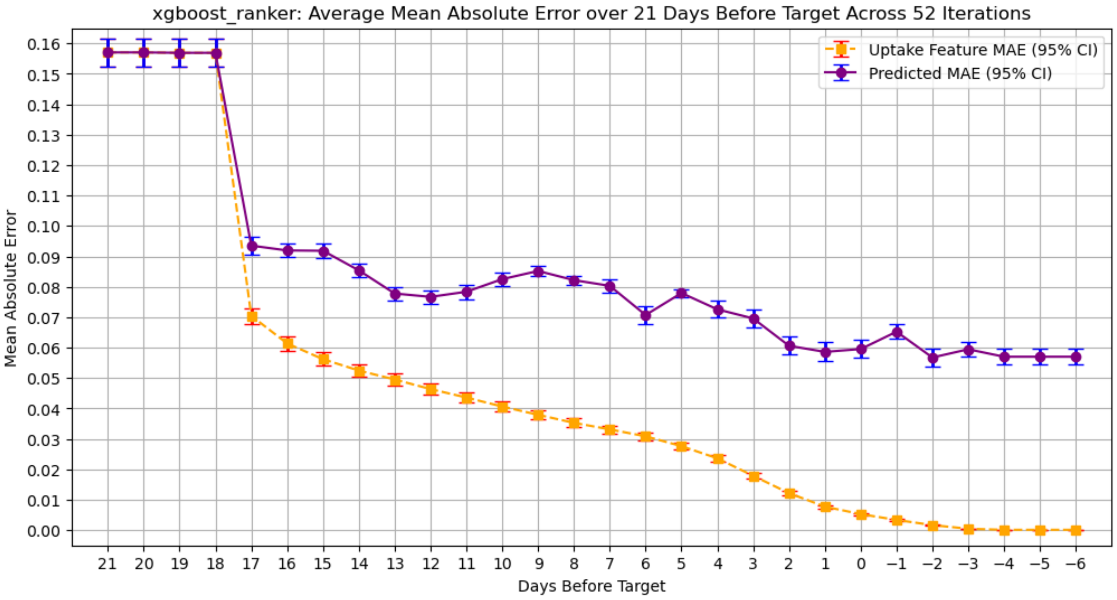

Figure A.17: MAE of xgboostranker52live from 1 to 8 weeks before target, across 52 iterations

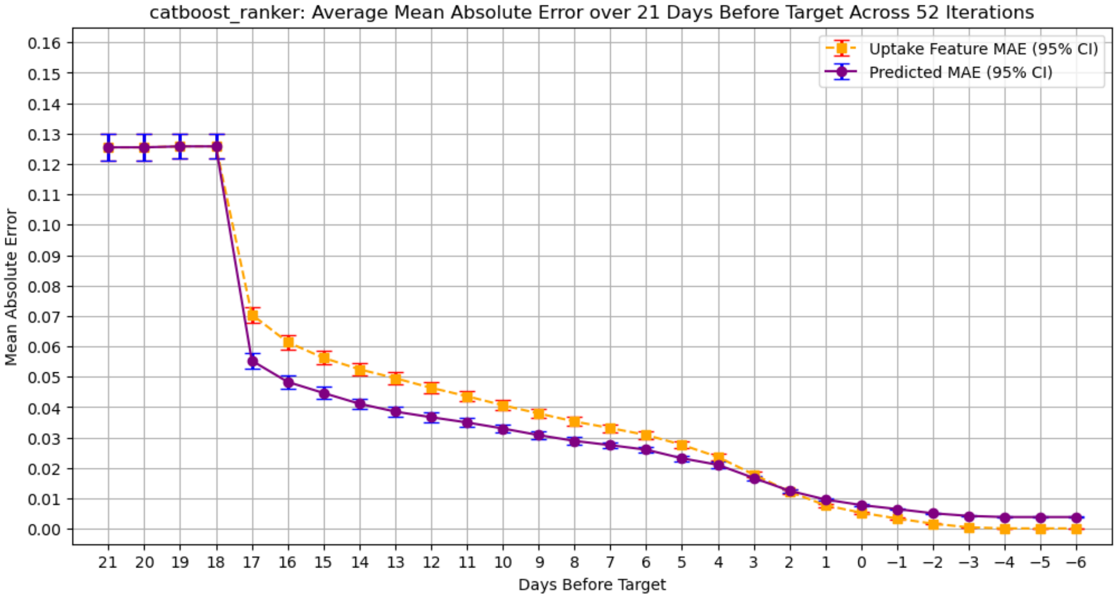

Figure A.18: MAE of cbranker52live from 1 to 8 weeks before target, across 52 iterations

A.3 Hypothesis Test Full Results

Lead_Day	Model	Mean	Std Dev	95% CI	T-statistic	P-value	Best Model
17	LGBM	0.051423	0.008581	(0.049033, 0.053812)	-12.804500	0.000000	LightGBM
	Linear	0.058322	0.009111	(0.055785, 0.060858)			
16	LGBM	0.045102	0.008215	(0.042814, 0.047389)	-7.493200	0.000000	LightGBM
	Linear	0.048758	0.008263	(0.046458, 0.051058)			
15	LGBM	0.041347	0.007623	(0.039225, 0.043470)	-6.001500	0.000000	LightGBM
	Linear	0.043694	0.007295	(0.041663, 0.045725)			
14	LGBM	0.038016	0.006463	(0.036216, 0.039815)	-5.619800	0.000001	LightGBM
	Linear	0.040294	0.006847	(0.038388, 0.042201)			
13	LGBM	0.035893	0.005698	(0.034307, 0.037480)	-5.859800	0.000000	LightGBM
	Linear	0.037764	0.005990	(0.036096, 0.039431)			
12	LGBM	0.033862	0.005539	(0.032319, 0.035404)	-3.975200	0.000222	LightGBM
	Linear	0.035218	0.005448	(0.033701, 0.036735)			
11	LGBM	0.031725	0.004729	(0.030409, 0.033042)	-3.750500	0.000452	LightGBM
	Linear	0.032963	0.005055	(0.031556, 0.034370)			
10	LGBM	0.029621	0.004812	(0.028282, 0.030961)	-3.349800	0.001528	LightGBM
	Linear	0.030793	0.004895	(0.029430, 0.032155)			
9	LGBM	0.028233	0.004023	(0.027113, 0.029353)	-2.224200	0.030591	LightGBM
	Linear	0.028925	0.004118	(0.027779, 0.030071)			
8	LGBM	0.026114	0.003968	(0.025010, 0.027219)	-1.145000	0.257543	Similar (LGBM)
	Linear	0.026455	0.003823	(0.025391, 0.027519)			
7	LGBM	0.024878	0.003336	(0.023950, 0.025807)	-0.853500	0.397363	Similar (LGBM)
	Linear	0.025094	0.003661	(0.024075, 0.026114)			
6	LGBM	0.023295	0.003342	(0.022364, 0.024225)	-0.071500	0.943275	Similar (LGBM)
	Linear	0.023311	0.003388	(0.022368, 0.024254)			
5	LGBM	0.020911	0.003278	(0.019999, 0.021824)	-0.098200	0.922150	Similar (LGBM)
	Linear	0.020933	0.003467	(0.019967, 0.021898)			
4	LGBM	0.018578	0.003412	(0.017628, 0.019529)	2.485200	0.016269	Linear
	Linear	0.018068	0.003639	(0.017055, 0.019081)			
3	LGBM	0.014144	0.002985	(0.013313, 0.014976)	0.783000	0.437260	Similar (Linear)
	Linear	0.014040	0.002989	(0.013200, 0.014880)			
2	LGBM	0.010142	0.002298	(0.009502, 0.010781)	0.989500	0.327073	Similar (Linear)
	Linear	0.010038	0.002316	(0.009393, 0.010683)			
1	LGBM	0.007107	0.001713	(0.006630, 0.007584)	2.139100	0.037235	Linear
	Linear	0.006956	0.001795	(0.006456, 0.007456)			
0	LGBM	0.005084	0.001296	(0.004723, 0.005445)	2.819500	0.006832	Linear
	Linear	0.004948	0.001294	(0.004588, 0.005308)			
-1	LGBM	0.003378	0.000891	(0.003130, 0.003626)	3.182000	0.002491	Linear
	Linear	0.003270	0.000952	(0.003005, 0.003535)			
-2	LGBM	0.001685	0.000696	(0.001492, 0.001879)	2.075000	0.043050	Linear
	Linear	0.001662	0.000684	(0.001472, 0.001853)			
-3	LGBM	0.000467	0.000262	(0.000394, 0.000541)	-	-	Similar
	Linear	0.000467	0.000262	(0.000394, 0.000541)			
-4/-5/-6	LGBM	0.000110	0.000116	(0.000078, 0.000143)	-	-	Similar
	Linear	0.000110	0.000116	0.000078, 0.000143)			

Table A.1: Results Summary: Full Dataset - LightGBM vs Linear (Days Before 17 to -6, $\alpha = 0.05$)

Lead_Day	Model	Mean	Std Dev	95% CI	T-statistic	P-value	Best Model
17	Piecewise	0.051423	0.008581	(0.049033, 0.053812)	-28.6657	0.000000	Piecewise
	Real-Time	0.070469	0.009361	(0.067863, 0.073075)			
16	Piecewise	0.045102	0.008215	(0.042814, 0.047389)	-24.9181	0.000000	Piecewise
	Real-Time	0.061381	0.008469	(0.059023, 0.063739)			
15	Piecewise	0.041347	0.007623	(0.039225, 0.043470)	-22.7567	0.000000	Piecewise
	Real-Time	0.056217	0.008163	(0.053944, 0.058489)			
14	Piecewise	0.038016	0.006463	(0.036216, 0.039815)	-23.5204	0.000000	Piecewise
	Real-Time	0.052469	0.007427	(0.050401, 0.054536)			
13	Piecewise	0.035893	0.005698	(0.034307, 0.037480)	-23.6154	0.000000	Piecewise
	Real-Time	0.049530	0.007040	(0.047570, 0.051490)			
12	Piecewise	0.033862	0.005539	(0.032319, 0.035404)	-23.0074	0.000000	Piecewise
	Real-Time	0.046376	0.006810	(0.044480, 0.048272)			
11	Piecewise	0.031725	0.004729	(0.030409, 0.033042)	-22.3747	0.000000	Piecewise
	Real-Time	0.043604	0.006455	(0.041807, 0.045401)			
10	Piecewise	0.029621	0.004812	(0.028282, 0.030961)	-23.4703	0.000000	Piecewise
	Real-Time	0.040640	0.006320	(0.038881, 0.042400)			
9	Piecewise	0.028233	0.004023	(0.027113, 0.029353)	-20.7689	0.000000	Piecewise
	Real-Time	0.037926	0.005646	(0.036354, 0.039498)			
8	Piecewise	0.026114	0.003968	(0.025010, 0.027219)	-21.0315	0.000000	Piecewise
	Real-Time	0.035284	0.005228	(0.033829, 0.036740)			
7	Piecewise	0.024878	0.003336	(0.023950, 0.025807)	-21.2097	0.000000	Piecewise
	Real-Time	0.033122	0.004814	(0.031781, 0.034462)			
6	Piecewise	0.023295	0.003342	(0.022364, 0.024225)	-21.6056	0.000000	Piecewise
	Real-Time	0.030848	0.004550	(0.029581, 0.032115)			
5	Piecewise	0.020911	0.003278	(0.019999, 0.021824)	-22.9151	0.000000	Piecewise
	Real-Time	0.027617	0.004383	(0.026397, 0.028838)			
4	Piecewise	0.018068	0.003639	(0.017055, 0.019081)	-16.6900	0.000000	Piecewise
	Real-Time	0.023614	0.004098	(0.022473, 0.024755)			
3	Piecewise	0.014040	0.002989	(0.013208, 0.014872)	-16.2729	0.000000	Piecewise
	Real-Time	0.017873	0.003423	(0.016920, 0.018826)			
2	Piecewise	0.009987	0.002564	(0.009041, 0.010933)	-10.2511	0.000000	Piecewise
	Real-Time	0.012895	0.002968	(0.012027, 0.013763)			
1	Piecewise	0.006956	0.001795	(0.006456, 0.007456)	-7.2354	0.000000	Piecewise
	Real-Time	0.007725	0.002072	(0.007148, 0.008302)			
0	Piecewise	0.004948	0.001294	(0.004588, 0.005308)	-4.8128	0.000014	Piecewise
	Real-Time	0.005275	0.001551	(0.004843, 0.005706)			
-1	Piecewise	0.003270	0.000952	(0.003005, 0.003535)	-3.0232	0.003907	Piecewise
	Real-Time	0.003400	0.001000	(0.003100, 0.003600)			
-2	Piecewise	0.001662	0.000684	(0.001472, 0.001853)	0.4113	0.682575	Similar (Real-Time)
	Real-Time	0.001661	0.000684	(0.001470, 0.001851)			
-3	Piecewise	0.000467	0.000262	(0.000394, 0.000541)	0.2479	0.805190	Similar (Real-Time)
	Real-Time	0.000467	0.000262	(0.000394, 0.000541)			
-4/-5/-6	Piecewise	0.000110	0.000116	(0.000078, 0.000143)	0.9934	0.325212	Similar (Real-Time)
	Real-Time	0.000110	0.000116	(0.000078, 0.000143)			

Table A.2: Results Summary: Full Dataset - Piecewise vs Real-Time Feature (Days Before 17 to -6, $\alpha = 0.05$)