Check-in Behavior as a Proxy for Customer Satisfaction: A Causal Inference and Machine Learning Approach to Restaurant Success

Term Paper

Submitted to
Dr. Keyvan Dehmamy
Faculty of Economics and Business Administration
Johann Wolfgang Goethe University
Frankfurt am Main

by
Ngoc Khanh Uyen Tran
Program: M.Sc. Management Science
Matriculation Number: 8518013
E-Mail: s0657641@stud.uni-frankfurt.de

Data Science and Marketing Analytics
Semester: Summer 2025

Abstract

This study examines check-in behavior as a proxy for customer satisfaction in restaurant businesses by applying causal inference theory and predictive modeling with machine learning algorithms. The study proposes a theoretical framework, in which customer satisfaction is a latent mediator between predictors and check-in behavior, indicating indirect measurement of customer satisfaction through observable behavioral data. With Yelp dataset of restaurants in Saint Louis from January 2020 to December 2021, nine machine learning algorithms were implemented and tuned with customized hyperparameter settings for binary classification, including Logistic Regression, Naive Bayes, k-Nearest Neighbors, Support Vector Machine, Decision Tree, Random Forest, Bagging, Boosting, and Neural Network, to predict customer's check-in behavior from static restaurant attributes, dynamic user-generated content, temporal patterns, and external weather conditions. Random Forest and Boosting are recognized as the most optimal models, with Random Forest algorithm demonstrating exceptional performance in targeting efficiency, whereas Boosting algorithm is particularly proficient for segment scoring capability. Causal interpretation from the variable analysis is converted into actionable insights for restaurant owners, including controllable interventions, strategic interventions and weather-responsive strategies to enhance customer satisfaction and, thus, achieve business success.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

SMOTE	Synthetic Minority Over-sampling Technique
TDL	Top Decile Lift
AUC	Area Under the ROC Curve
ROC	Receiver-operating Characteristic Curve
LG	Logistic Regression
NB	Naive Bayes
KNN	k-Nearest Neighbors
SVM	Support Vector Machine
DT	Decision Tree
RF	Random Forest

1 Introduction

Understanding customer satisfaction represents a fundamental challenge yet opportunity that require businesses to invest significant efforts onto, as it could directly impact sustainability and profitability of the organization (Anderson et al., 2004). Thus, in recent decades, the growth of technology advancements has also created new opportunities for firms to capture and enhance customer engagement through digital platforms and online communities. Specifically, within marketing analytics context, user-generated content and word-of-mouth have been rising as behavioral indicators that capture authentic customer responses without the limitations of traditional survey-based or self-reported approaches (Verhoef et al., 2009; Kim et al., 2009). However, there is still a gap for interpretability, in which factors that influence customer satisfaction are in need of causal understanding to go beyond just pure pattern recognition from machine learning algorithms.

Accordingly, this study aims to address the fundamental question: *"Which factors determine customer satisfaction in restaurant businesses?"*. Building on the causal inference perspective and causal model framework, the subsequent hypothesis is proposed: "*Restaurant check-in behavior is influenced by customer satisfaction, which is determined by static business attributes, dynamic user-generated content, temporal context, and external environmental conditions".* The study conceptualizes customer satisfaction as a latent mediator between predictors and check-in as target outcome, providing indirect measurement of customer satisfaction through observable behavioral data. This theoretical foundation guides the subsequent analysis and binary classifiers evaluation, translating predictive model insights into intervention recommendations through causal interpretation, providing a more comprehensive data-driven decision-making support for restaurant businesses.

2 Literature Review

2.1 Customer Satisfaction and Behavior in Hospitality Industry

Customer satisfaction in the hospitality industry, particularly restaurants, is influenced by multiple factors such as service quality, food quality, perceived value, and physical environment (Andaleeb & Conway, 2006; Stranjancevic & Bulatovic, 2015; Dwaikat et al., 2019). Customer satisfaction acts as a mediator between restaurant's perceived value and behavioral intentions, often through observable behaviors including repeat visitation, worth-of-mouth and recommendation, and digital platform participation (Tarn, 1999; Ha &

Jang, 2010). These findings underscore the importance of understanding and managing customer satisfaction to enhance service quality and customer engagement in the restaurant industry.

Table 1: Some existing studies on customer satisfaction in hospitality industry

Studies	Methodology	Key Findings
Stranjancevic and Bulatovic (2015)	Empirical survey and statistical methods	Restaurant guest satisfaction depends on factors such as staff, professionalism, speed of service, food quality, ambience, and comfort
Dwaikat et el. (2019)	Data collection with questionnaire of 386 responses from pizza restaurant customers in Nablus, and data analysis involving structural equational modeling	Customer perceived value has the strongest effect in customer satisfaction. Customer satisfaction positively affects behavioral intentions
Tarn (1999)	Data collection using self-reported questionnaires, and data analysis applying structural equation modeling	Customer satisfaction has the strongest effect in behavioral intentions. The effect if service quality on behavioral intentions is mediated though customer satisfaction

2.2 Causal Inference and Machine Learning in Marketing Analytics

Recent research explores the application of causal inference for interpreting predictive models in marketing analytics. For instance, Singh et al. (2023) propose a machine learning approach comparing causal inference modeling strategies in digital advertising, focusing on uplift modeling to estimate individual treatment effects. Khademi and Honavar (2020) introduce a method for interpreting "black box" of predictive models by estimating causal effects of inputs on outputs, demonstrating its effectiveness on deep neural networks. Similarly, Kumar and Ravi (2022) apply causal inference principles to analytical customer relationship management in banking and insurance, generating counterfactuals to explain model decisions. These studies collectively indicate the potential, as well as highlight the need for further development, of causal inference to enhance interpretability and understanding of predictive models in marketing and customer analytics (Parida, 2021).

Table 2: Some existing studies on causal inference and machine learning applications

Studies	Methodology	Key Findings
Khademi and Honavar (2020)	Estimating causal effects using variants of the Rubin Neyman potential outcomes framework on observational data and applying the approach on deep neural networks	The study develops method to interpret black box predictive models by estimating the causal effects of model inputs on the model output
Kumar and Ravi (2022)	Applying causal inference to address analytical customer relationship management problems with CausalNex, DiCE, and NOTEARS algorithm for DAGs	The study provides new approach to explainability of analytical customer relationship management and provision of valid counterfactuals to customers to enhance customer trust

Parida (2021)	Application of causal analysis in machine learning and deep learning models for explaining input feature influence on neural outputs	Causal analysis enhances predictive and prescriptive models by identifying cause-effect relations and feature dependencies. The study highlights the need for further development to integrate causal inference processes into machine learning applications

3 Theoretical Framework

3.1 Check-in as a Proxy for Customer Satisfaction

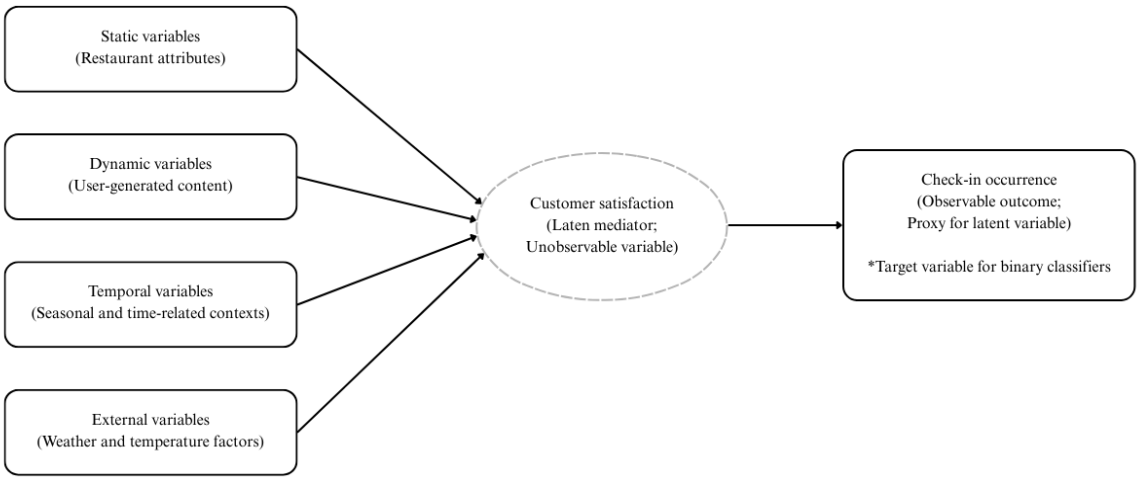
Instead of survey-based or sentiment analysis of customer satisfaction, this study utilizes predictive modeling with behavioral data as target event, specifically check-in occurrence, representing a proxy indicator for customer experience and satisfaction. A check-in represents a voluntary action by customers to mark their visit to a place, often in real time and on-location, which tends to resonate with positive affect, satisfaction with the experience, and potential intentions to re-engage with the entity. This study refers to latent variable theory, where customer satisfaction is conceptualized as an unobservable construct inferred through observable behaviors (Bagozzi & Yi, 1988). The theoretical foundation was inspired from the potential outcomes framework (Rubin, 1974), positioning check-in frequency as a downstream behavioral indicator of upstream satisfaction drivers. This approach also aligns with service-dominant logic (Vargo & Lusch, 2004), which indicates observable customer engagement as a consequence of service experience quality, therefore is recognized as satisfaction measurement (Fornell et al., 1996).

3.2 Causal Assumptions: Research Hypothesis

This study applies a causal inference perspective to understand which factor determines check-in behavior by modeling customer satisfaction as a latent variable between predictors and check-in outcomes. The core causal hypothesis is: *Restaurant check-in behavior is influenced by customer satisfaction, which is determined by static business attributes, dynamic user-generated content, temporal context, and external environmental conditions.*

Accordingly, this study structures the theoretical framework with four causal pathways, which are recognized as predictors and treatment variables, to customer satisfaction: (1) static variables, which represents fixed restaurant attributes; (2) dynamic variables, which represents user-generated content that are subject to change over time; (3) temporal variables, which represents seasonal and time-related patterns; and (4) external variables, which represents weather and temperature factors.

Figure 1: Causal assumptions - Research hypothesis



Under the causal perspective, customer satisfaction is treated as a latent mediator between these observable predictors and behavioral outcomes. Thus, the check-in occurrence acts as a proxy for customer satisfaction, enabling the empirical estimation of underlying causal mechanisms.

4 Data

In this study, different datasets were acquired from Yelp, an online crowd-sourced information platform specializing in user-generated content and reviews ((Luca, 2011), covering information on restaurants in the city of Saint Louis in from January 2020 to December 2021. The variables range from business features, user information, review characteristics, and behavioral indicators. Furthermore, weather variables were also collected as external data for more comprehensive analysis and implementation of machine learning models.

4.1 Data Preprocessing

Initial data preprocessing involved loading the daily-level user-generated information filtered to exclude business-specific information. Only the first nine columns, corresponding to non-business attributes, were retained to avoid missing values in business-specific variables. Some variables were stored as logical vectors or *integer64* type, which were converted to the standard integer type to ensure compatibility across engineering functions and machine learning algorithms. To enable demographic analysis, a parallelized procedure was implemented to infer gender counts from reviewer names from the column *cum_u_name* using the *gender* and *genderdata* libraries. For each entry, the number of male and female reviewers was computed and stored in two new columns. Subsequently, the original reviewer names column was removed. To analyze temporal data later, the tip

date column was converted to the standard date format and, therefore, the original date column was removed.

Then, business-level data were loaded separately and preprocessed to handle missing values. Variables of type *integer64* again were converted to the standard integers. The *business_id* column, a factor with more than 50 levels, was excluded from imputation processes to avoid incompatibility with the *mice* imputation package. Missing values in the business data were inspected using the *md.pattern* function, and imputation was performed using the *mice* package for columns with incomplete data. Character-type variables were converted to factors to be utilized for categorical analyses. For the business category variable, the number of categories was limited to four: Asian, American, Mexican, and Others, using pattern matching and reclassification, with all uncategorized entries assigned to "Others". Certain variables, such as *n_photos* and *cum_max_u_elite*, were found to have missing values that, upon inspection, were deemed to represent zeros. These missing values were accordingly replaced by zero, indicating the absence of photos and elite reviewer contributions rather than actual missing information. Lastly, the imputed business dataset was merged with the daily-level user-generated information data using the *business_id* key.

Beside the datasets from Yelp, this study also acquired external data, specfically weather-related information queried from the National Eceanic and Atmospheric Administration (NOAA) database using a parallelized grid-search approach, matching the geographical and temporal scope of the Yelp dataset. Daily averages of key weather variables were computed and merged with the main dataset by date and geographic proximity. For entries with missing weather data, the procedure iteratively searched for the nearest available weather station, ensuring maximal coverage. Additional temporal features, such as indicators for weekends and calendar quarters, were engineered and added as factor variables.

For categorical variables, they were transformed into dummy variables for statistical analysis later. Firstly, *business_cat* variable was expanded into separate columns for each specific restaurant category using one-hot encoding, allowing the model to interpret the presence or absence of each business type. Secondly, *business_open* variable was converted to a logical variable and then to two dummy variables, *business_open_TRUE* and *business_open_FALSE*, indicating whether the business was open or closed. Thirdly, *WE* variable was transformed into logical and then dummy variables, *WE_TRUE* and *WE_FALSE*, indicating weekends and non-weekend days respectively. Finally, *Quarter* variable was one-hot encoded, creating separate columns for each specific calendar quarter. After the

transformation, the original categorical variables were removed from the dataset, leaving only their corresponding dummy variables.

For prediction tasks, the main outcome for binary classification is whether a check-in occurred. The raw variable *ch_in*, a variable of integer count of check-ins, was transformed into a categorical variable *ch_in_string*, where entries with one or more check-ins were labeled as "ch_in", and entries with zero check-ins were labeled as "Noch_in". The reference level for modeling was set as "Noch_in", which is standard practice when evaluating performance on minority classes.

A detailed overview of all variables, grouped by category, with their description and statistical analysis are presented in the following sections.

4.2 Static Data: Business-related Variables

Static data contains variables that describe the relatively unchanging characteristics of a business. These variables typically do not vary over short time frames and represent the physical attributes of each business.

Table 3: List of static variables

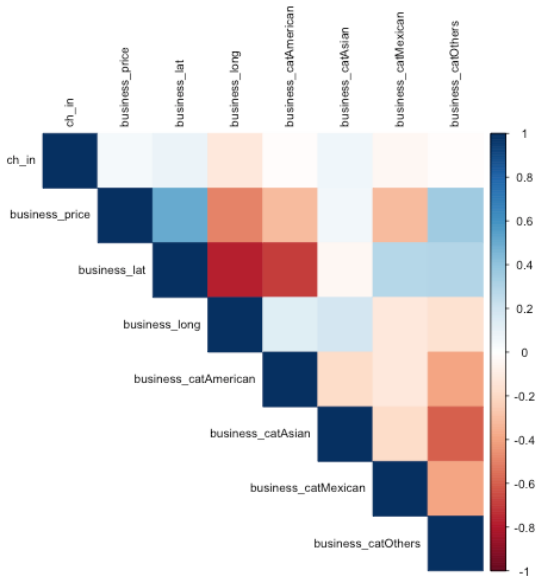
Variable	Type	Description
business_id	Character	Unique business identifier
business_open_TRUE	Logical	Business is open
business_open_FALSE	Logical	Business is not open
business_price	Integer	Business price category
business_lat	Numeric	Latitude coordinate of business
business_long	Numeric	Longitude coordinate of business
business_park	Factor	Business has parking space
business_catAmerican	Numeric	Business category is American
business_catAsian	Numeric	Business category is Asian
business_catMexican	Numeric	Business category is Mexican
business_catOthers	Numeric	The other business categories

The static variables in the dataset are largely independent. Most businesses falling into the lower price category (mean = 1.44) and the majority, approximately 56%, classified as "Others" in business category. Latitude and longitude values are tightly clustered, reflecting the geographic focus on Saint Louis area, with just the small standard deviations indicate minimal spatial dispersion within the sample. “Asian” category contains about 22% of the sample, whereas “American” and “Mexican” categories each include around 11% of businesses. Correlation analysis shows that while most static variables are weakly correlated, there is a positive correlation between business latitude and longitude, and moderate associations between some business categories and location coordinates. The business categories themselves are mutually exclusive and, therefore, strongly negatively correlated.

Table 4: Descriptive statistics of static variables and the target variable

Statistic	N	Mean	St. Dev.	Min	Median	Max
ch_in	6,579	0.020	0.139	0	0	1
business_price	6,579	1.444	0.497	1	1	2
business_lat	6,579	38.648	0.025	38.599	38.656	38.683
business_long	6,579	-90.295	0.083	-90.475	-90.308	-90.192
business_catAmerican	6,579	0.111	0.314	0	0	1
business_catAsian	6,579	0.222	0.416	0	0	1
business_catMexican	6,579	0.111	0.314	0	0	1
business_catOthers	6,579	0.556	0.497	0	1	1

Figure 2: Correlation map of static variables and the target variable



4.3 Dynamic Data: User-generated Content and Behavioral Variables

Dynamic variables involve variables that capture user behavior and activity over time. These variables are subject to frequent change and reflect real-time engagement.

Table 5: List of dynamic variables

Variable	Type	Description
ch_in/ch_in_string	Integer/Factor	Check-in indicator
cum_n_tips	Integer	Cumulative number of tips
cum_max_friends	Integer	Cumulative users' friends
cum_max_u_elite	Integer	Cumulative the elite year of users
cum_max_us_fans	Integer	Cumulative users' fans
cum_max_us_tip	Integer	Cumulative tips by users
male	Integer	Count of male users
female	Integer	Count of female users
n_photo	Integer	Number of photos

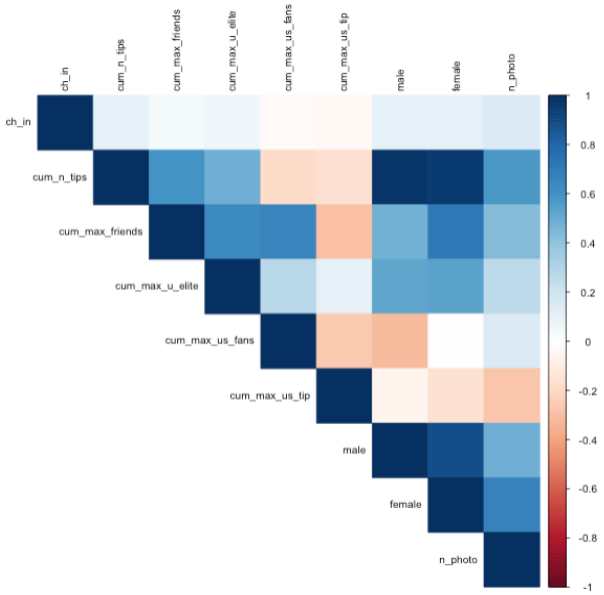
Dynamic variables show substantial heterogeneity on behavioral patterns. For instance, the typical restaurant in the sample receives around 25 tips, but some receive up to four times this amount, reflecting wide variability in user engagement. The number of elite users per restaurant is relatively stable, but the number of fans and tips by users can vary by more than 80 times across observations. Gender participation is balanced, with the number of

male and female users differing by less than 10%. Correlation analysis indicates that dynamic variables are strongly correlated, with businesses popular in one dimension tend to be also popular across others. In contrast, the check-in variable shows weak association with other dynamic variables, suggesting that even check-in can be interpreted as high customer engagement, however, high user engagement may not increase check-ins.

Table 6: Descriptive statistics of dynamic variables and the target variable

Statistic	N	Mean	St. Dev.	Min	Median	Max
ch_in	6,579	0.020	0.139	0	0	1
cum_n_tips	6,579	25.530	25.764	7	18	100
cum_max_friends	6,579	1,901.889	2,156.723	265	431	5,522
cum_max_u_elite	6,579	8.111	2.961	4	8	13
cum_max_us_fans	6,579	158.556	257.040	6	9	740
cum_max_us_tip	6,579	144.111	235.368	7	96	800
male	6,579	9.544	9.757	1	7	36
female	6,579	8.708	9.636	1	6	37
n_photo	6,579	10.889	12.088	0	3	33

Figure 3: Correlation map of dynamic variables and the target variable



4.4 Temporal Data: Time-related Variables

Temporal variables contain the time stamp for each observation, including weekend and quarters, indicating whether check-ins are associated with seasonal or calendar effects.

Table 7: List of temporal variables

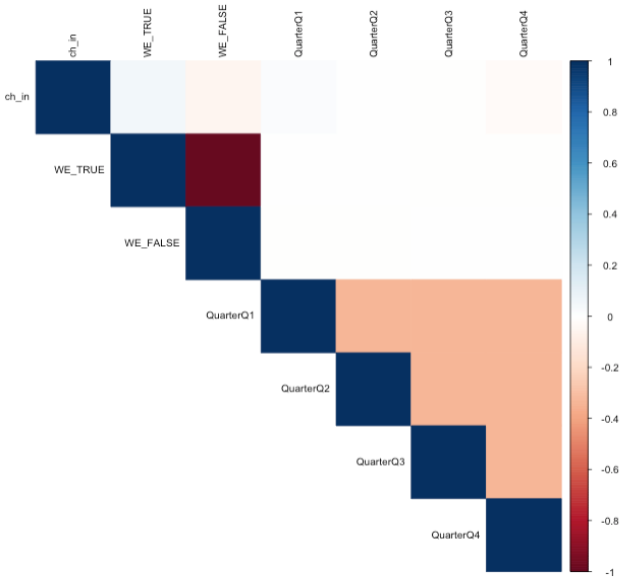
Variable	Type	Description
WE_TRUE	Numeric	Day is weekend
WE_FALSE	Numeric	Day is not weekend
QuarterQ1	Numeric	Quarter 1 (January - March)
QuarterQ2	Numeric	Quarter 2 (April - June)
QuarterQ3	Numeric	Quarter 3 (July - September)
QuarterQ4	Numeric	Quarter 4 (October - December)

Descriptive statistics for temporal variables indicate that approximately 28.5% of observations are weekends, while the distribution across calendar quarters is relatively balanced, with each quarter representing about 25% of the data. Correlation analysis highlights the expected strong negative relationship between weekend and non-weekend indicators, as well as mutually exclusive patterns among quarterly dummies.

Table 8: Descriptive statistics of temporal variables and the target variable

Statistic	N	Mean	St. Dev.	Min	Median	Max
ch_in	6,579	0.020	0.139	0	0	1
WE_TRUE	6,579	0.285	0.451	0	0	1
WE_FALSE	6,579	0.715	0.451	0	1	1
QuarterQ1	6,579	0.248	0.432	0	0	1
QuarterQ2	6,579	0.249	0.432	0	0	1
QuarterQ3	6,579	0.252	0.434	0	0	1
QuarterQ4	6,579	0.252	0.434	0	0	1

Figure 4: Correlation map of temporal variables and the target variable



4.5 External Data: Weather Variables

External variables contain contextual information about weather and temperature conditions, which may influence consumer behavior. These features are linked to each observation by date and location, providing an environmental background to customer check-ins.

Table 9: List of external variables

Variable	Type	Description
PRCP	Numeric	Precipitation amount
SNOW	Numeric	Snow fall amount
SNWD	Numeric	Snow depth
TMAX	Numeric	Maximum temperature
TMIN	Numeric	Minimum temperature
TOBS	Integer	Observed temperature
TOBS_1	Integer	Observed temperature at hour 1
TOBS_2	Integer	Observed temperature at hour 2

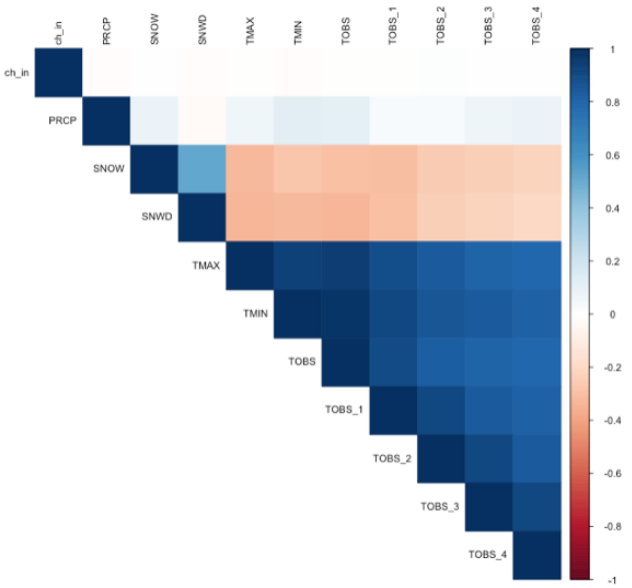
TOBS_3	Integer	Observed temperature at hour 3
TOBS_4	Integer	Observed temperature at hour 4

Weather-related external variables show high variability. For instance, precipitation ranges from 0 to over 700 units, whereas the median is less than 1, indicating though most days are dry, rare events still contribute to a highly skewed distribution. Snowfall and snow depth are zero for the majority of observations, and only extreme in few cases. Temperature variables display a wide range, with negative minimums reflecting rare cold events, and strong consistency across different hours in a same day. Correlation analysis indicates that temperature measures are positively correlated. Snowfall and snow depth are also strongly correlated with each other and moderately with precipitation, reflecting their shared dependence on weather events.

Table 10: Descriptive statistics of external variables and the target variable

Statistic	N	Mean	St. Dev.	Min	Median	Max
ch_in	6,579	0.020	0.139	0	0	1
PRCP	6,579	31.949	76.481	0.000	0.500	720.750
SNOW	6,579	0.691	5.194	0.000	0.000	121.000
SNWD	6,576	1.805	13.228	0.000	0.000	180.000
TMAX	6,552	193.086	106.020	-161.000	204.250	372.000
TMIN	6,546	90.291	99.786	-200.000	80.500	267.000
TOBS	6,579	112.266	98.758	-200	111	300
TOBS_1	6,098	111.193	98.924	-200	106	300
TOBS_2	6,068	111.446	98.602	-200	106	289
TOBS_3	6,027	111.555	98.309	-200	106	300
TOBS_4	6,018	111.789	98.544	-200	106	300

Figure 5: Correlation map of external variables and the target variable



5 Model Training and Hyperparameter Tuning

5.1 Feature Engineering

Prior to model training and according to the above descriptive statistics of all variable groups, several columns were excluded from the dataset due to their low predictive power or redundancy, including business identifiers, date, number of photos, business open and not open, presence of parking lot, snow depth, maximum and minimum temperature, and observed temperature at specific hours. Overall, after preprocessing and feature engineering, the final dataset contained 6579 observations and 25 variables.

The dataset was then separated into training and evaluation sets, with a split ratio of 75:25, as well as was randomly shuffled to eliminate any order effects. The random sampling preserves the representativeness of both subsets and reduces sampling bias. Subsequently, there are 4934 samples in the training set, and 1645 samples in the evaluation set.

All categorical variables, including the outcome variable *ch_in_string*, were transformed into dummy variables using one-hot encoding. This step was essential for compatibility with machine learning algorithms and for subsequent application of the SMOTE algorithm. With check-in events being relatively rare, with only 1.9%, indicating class imbalance issue within target variable, the Synthetic Minority Over-sampling Technique (SMOTE) technique was applied to generate synthetic samples for the minority class based on feature space similarities, ensuring the models were not biased toward the majority class. Furthermore, for heuristic machine learning methods, excluding logistic regression, variables were normalized to a [0,1] range. This transformation standardized the scale of the input features, preventing variables with larger ranges from excessively influencing the models.

5.2 Performance Metrics and Cross-validation Setting

For marketing analytics, the evaluation of predictive models for training and fine-tuning requires metrics that translate statistical performance into actionable insights. Therefore, this study considers three primary metrics: TDL, GINI coefficient, AUC. High values in these metrics mean that marketing resources can be focused on the most promising customer segments, maximizing return-on-investment. Top Decile Lift (TDL) measures how effectively a model identifies high-probability check-in customers within the top 10% of predicted probabilities compared to random selection (Berry & Linoff, 2004). Meanwhile, the GINI coefficient, related to the Area Under the ROC Curve (AUC), quantifies the model's ability to distinguish between positive and negative classes, which are check-in

and non-check-in, with coefficient values range from 0 (random performance) to 1 (perfect discrimination). (Hanley & McNeil, 1982; Fawcett, 2006).

Furthermore, this study also considers three secondary metrics: F1-score, Precision, Recall, to measure prediction efficiency, correct prediction coverage and the costs of prediction errors in marketing contexts, respectively (Sokolova & Lapalme, 2009). Additional, three tertiary metrics: Accuracy, Specificity, Computational time are considered less important because they can be misleading in an imbalanced dataset, in which high accuracy can be achieved by simply predicting the majority class and Specificity focuses on correctly identifying the negative class while the primary goal of this study is to correctly identify the minority class, such as users who check-in.

The 10-fold cross-validation was also implemented using stratified folds to preserve class proportions. Cross-validation was configured to compute class probabilities and retain final predictions, thereby supporting robust model evaluation and comparison.

5.3 Machine Learning Algorithms

In this study, nine machine learning algorithms were implemented for binary classification, including: (1) Linear model: Logistic regression, (2) Probabilistic model: Naive Bayes, (3) Instance-based model: k-Nearest Neighbors, (4) Margin-based model: Support Vector Machine, tree-based and ensemble methods: (5) Decision Tree, (6) Random Forest, (7) Bagging, (8) Boosting, (9) Neural Network.

5.3.1 Logistic Regression

Logistic regression (LG) models the probability of binary outcomes through the logistic function, with the predicted probabilities bounded between 0 and 1. The algorithm estimates coefficients β through the log-odds of the target variable follows a linear relationship (Hosmer et al., 2013). In this study, β represents the probability of check-in occurrence: $logit(P) = log(P/(1-P)) = \beta_0 + \beta_1 x_1 + ... + \beta_k x_k$.

Two LG models were implemented to evaluate the predictive power and select the best fine-tuned model. The base model applied standard maximum likelihood estimation without penalty terms. The tuned model applied elastic net regularization through the *glmnet* method, combining L1 (LASSO) and L2 (Ridge) penalties to address potential multicollinearity and perform automatic feature selection. The hyperparameter grid chose the combination of parameter $\alpha = 2.0$ and regularization strength $\lambda = 0.05$.

Table 11: Performance metrics of Logistic Regression models

Metrics	LG Base Model	LG Tuned Model
TDL	3.343	3.343
GINI	0.696	0.727
AUC	0.848	0.863
F1	97.728	86.015
Precision	97.880	98.949
Recall	97.576	76.072
Accuracy	95.562	75.805
Specificity	5.556	63.889
Time	146.321	17.550

There are obvious trade-offs between the base and tuned LG models across key metrics. Both models achieved TDL = 3.343, indicating equivalent targeting efficiency where marketing campaigns would reach 3.3 times more engaged customers than random selection. The tuned model demonstrates superior discrimination capability with GINI coefficient (0.727 vs. 0.696) and AUC (0.863 vs. 0.848), reflecting better overall ranking of customers by engagement probability. However, F1-scores decreased dramatically (97.728 vs. 86.015), as recall dropped significantly (97.576% to 76.072%), represents significant missed opportunities for customer targeting, while high precision maintained (97.880 vs. 98.949). Given marketing objectives prioritizing customer satisfaction prediction, the base LG model was chosen as the better fine-tuned model.

5.3.2 Naive Bayes

Naive Bayes (NB) is a probabilistic classifier based on Bayes' theorem with the "naive" assumption of conditional independence between features given the class label (Zhang, 2004). The algorithm calculates the probability of check-in occurrence using the formula: $P(\text{check-in}|\text{variables}) = P(\text{variables}|\text{check-in}) \times P(\text{check-in}) / P(\text{variables})$.

Two NB models were implemented to evaluate the predictive power of hyperparameter optimization. The base model applied parameters with standard Gaussian assumptions, while the tuned model applied Laplace smoothing $(\alpha = 10)$ to handle zero probabilities, kernel density estimation for improved continuous variable modeling, and bandwidth adjustment $(adjust = 1)$ for density estimation refinement.

Table 12: Performance metrics of Naive Bayes models

Metrics	NB Base Model	NB Tuned Model
TDL	3.065	3.065
GINI	0.762	0.762
AUC	0.881	0.881
F1	69.818	69.818
Precision	99.769	99.769
Recall	53.698	53.698
Accuracy	54.590	54.590

Specificity	94.444	94.444
Time	1.136	0.529

Performance evaluation reveals identical results across two NB models, indicating that hyperparameter tuning provided no improvement over base settings. Both models achieve moderate targeting efficiency (TDL = 3.065) and strong discrimination capability (GINI = 0.762, AUC = 0.881). However, F1-scores is relatively low (69.818) due to poor recall performance (53.698%), indicating the model's prediction bias that identifies only half of actual check-ins. The high precision (99.769%) and specificity (94.444%) suggest NB performs well at avoiding false positives but struggles with customer engagement identification. Therefore, the base NB model was selected as the better fine-tuned model.

5.3.3 k-Nearest Neighbors

k-Nearest Neighbors (KNN) is a non-parametric, instance-based learning algorithm that classifies observations based on the majority class among the k nearest neighbors in the feature space (Cover & Hart, 1967). The algorithm computes distances between test instances and training observations, typically using Euclidean distance: $d(x,y) = \sqrt{\Sigma(xi - yi)^2}$ Then, the probability of check-in occurrence is assigned as: *P(check-in|x) = (number of check-in neighbors) / k.*

Two KNN implementations were evaluated to determine optimal neighborhood size. The base model applied cross-validation to automatically select the optimal k value from a default range. The tuned model applied a fixed $k = 25$ neighbors based on grid search results, balancing bias-variance trade-offs. Feature normalization was applied to both models to ensure equal contribution of variables with different scales.

Table 13: Performance metrics of KNN models

Metrics	KNN Base Model	KNN Tuned Model
TDL	2.508	3.343
GINI	0.287	0.485
AUC	0.644	0.743
F1	95.705	94.133
Precision	98.044	98.374
Recall	93.474	90.242
Accuracy	91.793	88.997
Specificity	16.667	33.333
Time	6.150	2.122

The tuned model performs better across key marketing metrics, achieving significant improvements in targeting efficiency (TDL: 2.508 to 3.343) and discrimination capability (GINI: 0.287 to 0.485, AUC: 0.644 to 0.743), indicating better customer ranking for marketing campaigns. Despite a slight drop in F1-score (95.705 to 94.133), which is due the

decrease in recall (93.474% to 90.242%), precision remains consistently high (98.044% vs 98.374%), suggesting effective false positive prediction. Thus, the tuned KNN model was selected as the better fine-tuned model.

5.3.4 Support Vector Machine

Support Vector Machine (SVM) implements decision boundaries by maximizing the margin between classes while minimizing classification errors. The algorithm applies radial basis function kernel to transform data into higher-dimensional space where linear separation is possible. The decision function for target variable is defined as: $f(x) = sign(\Sigma \alpha i\ yi\ K(xi, x) + b)$, where support vectors (determined by non-zero αi) define the boundary and the regularization parameter C controls the trade-off between margin maximization and classification accuracy (Cortes & Vapnik, 1995; Scholkopf & Smola, 2002).

Two SVM implementations were evaluated to assess hyperparameter optimization effects. The base model applied cross-validation to automatically select optimal cost parameter C and kernel bandwidth γ from default ranges. The tuned model employed fixed hyperparameters $(C = 2, \sigma = 0.2)$ derived from preliminary grid search for balanced model complexity and generalized performance.

Table 14: Performance metrics of SVM models

Metrics	SVM Base Model	SVM Tuned Model
TDL	3.343	2.229
GINI	0.705	0.254
AUC	0.852	0.627
F1	92.667	95.165
Precision	98.394	98.151
Recall	87.570	92.356
Accuracy	86.444	90.821
Specificity	36.111	22.222
Time	88.445	19.806

Performance evaluation shows the lower predictive power of the tuned model compared with the base model in many key metrics, including targeting efficiency (TDL: 3.343 to 2.229) and discrimination capability (GINI: 0.705 to 0.254, AUC: 0.852 to 0.627). While F1-score improvements (92.667 to 95.165) suggest better overall classification with enhanced recall (87.570% to 92.356%), these gains are overweighted by the significant loss in customer ranking ability essential for marketing applications. Given marketing objectives and the relatively good performance even without advanced tunning, the base SVM model was selected as the better fine-tuned model.

5.3.5 Decision Tree

Decision Tree (DT) is a predictive model using recursive binary partitioning, creating hierarchical rule-based structures that segment the feature space based on optimal split criteria. The algorithm selects splits that maximize information gain or minimize impurity measures such as Gini impurity, where *pi* represents the proportion of each class at a node (Breiman et al., 1984; Quinlan, 1986): $Gini = 1 - \Sigma pi^2$. For check-in prediction, each leaf node provides probability estimates, enabling interpretable decisions in marketing: *P(check-in|leaf) = (check-in instances in leaf) / (total instances in leaf)*.

Two DT models were implemented with different hyperparameter settings. The base model applied conditional inference trees *(ctree)* with unbiased variable selection and automatic stopping criteria. The tuned model applied recursive partitioning *(rpart)* with complexity parameter grid search *(cp = 0.0001 to 0.5)*, minimum split requirement *(minsplit = 1)*, and maximum depth constraint *(maxdepth = 5)* to balance model complexity and interpretability while preventing overfitting.

Table 15: Performance metrics of Decision Tree models

Metrics	DT Base Model	DT Tuned Model
TDL	1.950	2.508
GINI	0.367	0.675
AUC	0.684	0.838
F1	95.967	93.938
Precision	98.117	98.171
Recall	93.909	90.056
Accuracy	92.280	88.632
Specificity	19.444	25
Time	8.461	3.687

The tuned model demonstrates better performance across key metrics, including targeting efficiency (TDL: 1.950 to 2.508) and discrimination capability (GINI: 0.367 to 0.675, AUC: 0.684 to 0.838), indicating better ranking for customer targeting. Despite a slight descrease in F1-score (95.967 to 93.938) due to recall (93.909% to 90.056%), precision remains consistently high (98.117% vs 98.171%). Therefore, the tuned DT model was selected as the better fine-tuned model.

5.3.6 Random Forest

Random Forest (RF) implements ensemble predictions by aggregating multiple decision trees trained on bootstrapped samples with feature randomization, where each tree considers only randomly selected features from the total feature set at each node split (Breiman, 2001). The check-in probability aggregates individual tree predictions, where *B* represents the number of trees: $P(check\text{-}in|x) = (1/B) \Sigma P(check\text{-}in|x, tree\_b)$.

Two RF models were implemented to evaluate hyperparameters' predictive performance. The base model applied parallel random forest *(parRF)* with the best combination from default parameters. The tuned model applied the ranger implementation with hyperparameter combination: feature subset size *(mtry = 2 to √features)*, split criteria *(gini and extratrees)*, minimum node size *(1, 5, 10)*, fixed tree count *(num.trees = 100)*, maximum depth constraint *(max.depth = 10)*, and impurity-based variable importance calculation.

Table 16: Performance metrics of Random Forest models

Metrics	RF Base Model	RF Tuned Model
TDL	2.508	3.901
GINI	0.700	0.729
AUC	0.850	0.864
F1	97.790	97.175
Precision	97.943	98.161
Recall	97.638	96.209
Accuracy	95.684	94.529
Specificity	8.333	19.444
Time	145.047	31.856

The tuned model performs significantly better in targeting efficiency (TDL: 2.508 to 3.901), representing the highest lift performance observed, while maintaining strong discrimination capability (GINI: 0.700 to 0.729, AUC: 0.850 to 0.864). Though F1-score slightly declined (97.790 to 97.175) due to recall (97.638% to 96.209%), precision increased slightly (97.943% to 98.161%), along with doubled specificity (8.333% to 19.444%) which indicates improved classification balance. Thus, the tuned RF model was selected as the better fine-tuned model.

5.3.7 Bagging

Bagging (Bootstrap Aggregating) implements ensemble predictions by training multiple decision trees on bootstrapped samples of the training data and averaging their outputs. The algorithm generates B bootstrap samples through random sampling with replacement, trains individual trees on each sample, then combines predictions as: $P(\text{check-in}|x) = (1/B) \Sigma P(\text{check-in}|x, \text{tree\_b})$. Unlike Random Forest, bagging considers all features at each split, focusing purely on variance reduction through ensemble averaging rather than feature randomization (Breiman, 1996; Bühlmann & Yu, 2002).

Two Bagging models were implemented, with the base model employed tree bagging *(treebag)* with standard bootstrap sampling and automatic ensemble size selection, while the tuned model sets the number of bootstrap aggregations *(nbagg = 100)* to increase ensemble diversity and stability while maintaining computational feasibility. Both models utilized unpruned decision trees as base learners to maximize individual tree variance.

Table 17: Performance metrics of Bagging models

Metrics	Bagging Base Model	Bagging Tuned Model
TDL	3.065	2.786
GINI	0.340	0.509
AUC	0.670	0.755
F1	97.441	97.600
Precision	97.868	97.875
Recall	97.017	97.328
Accuracy	95.015	95.319
Specificity	5.556	5.556
Time	17.966	45.939

Performance evaluation shows that the tuned model has an improved discrimination capability (GINI: 0.340 to 0.509, AUC: 0.670 to 0.755), however, the targeting efficiency decreased (TDL: 3.065 to 2.786). Even though F1-score, precision, recall, and accuracy all slightly improved around 0.02%, these increases are not sufficient to overweight the performance decline in TDL. Given the trade-offs, the base Bagging model was selected as the better fine-tuned model.

5.3.8 Boosting

Boosting implements ensemble predictions through sequential learning, where each subsequent model corrects errors made by previous models in the ensemble. The algorithm iteratively builds weak learners, with each iteration focusing on previously misclassified observations through adaptive weight adjustment. This sequential error correction approach combines weighted contributions, where *ht* represents individual weak learners and αt denotes their respective weights based on training performance (Freund & Schapire, 1997; Hastie et al., 2009): $P(\text{check-in}|x) = \Sigma\, \alpha t\, ht(x)$.

Two Boosting models were implemented using different hyperparameter settings. The base model applied *blackboost* with regression trees as base learners and automatic hyperparameter selection through cross-validation. The tuned model applied extreme gradient boosting *(xgbTree)* with optimized parameters: number of rounds *(nrounds = 150)*, maximum tree depth *(max_depth = 2)*, learning rate *(eta = 0.1)*, regularization parameter *(gamma = 0)*, feature sampling *(colsample_bytree = 1)*, minimum child weight *(min_child_weight = 1)*, and instance sampling *(subsample = 1)* to balance performance and overfitting prevention.

Table 18: Performance metrics of Boosting models

Metrics	Boosting Base Model	Boosting Tuned Model
TDL	3.343	3.065
GINI	0.749	0.761
AUC	0.875	0.880
F1	92.480	96.526

Precision	98.524	98.137
Recall	87.135	94.966
Accuracy	86.140	93.313
Specificity	41.667	19.444
Time	84.486	4.530

Performance evaluation shows trade-offs between the base and tuned models, with increased discrimination capability (GINI: 0.749 to 0.761, AUC: 0.875 to 0.880) and decreased targeting efficiency (TDL: 3.343 to 3.065). The tuned model achieves better F1-score (92.480 to 96.526) due to improved recall (87.135% to 94.966%), while maintaining relatively high precision (98.524% vs 98.137%), indicating better identification of check-in customers. However, the decline in targeting efficiency represents reduced marketing campaign effectiveness despite overall classification improvements. Therefore, the base Boosting model was selected as the better fine-tuned model.

5.3.9 Neural Network

Neural Network (NNet) models complex non-linear relationships through interconnected layers of artificial neurons that transform input features via weighted connections and activation functions. The algorithm propagates information forward through hidden layers using the transformation where σ represents the activation function, W denotes weight matrices, and b represents bias vectors (Ripley, 1996): $z = \sigma(Wx + b)$. For binary classification of check-in prediction, the output layer typically applies sigmoid activation to produce probabilities enabling the network to capture complex feature interactions and non-linear patterns in customer behavior data through backpropagation learning: $P(check\text{-}in|x) = 1/(1 + e^{(-z\_output)})$ (Bishop, 1995).

Two NNet models were implemented, with the base model applied an architecture with two hidden layers *(layer1 = 5, layer2 = 5, layer3 = 0)* using multilayer perceptron *(mlpML)*. The tuned model applied an expanded architecture *(layer1 = 30, layer2 = 2, layer3 = 0)* with increased first-layer capacity to capture feature interactions, decreased second layer for dimensionality reduction, and trained with extended iterations *(maxit = 200)*.

Table 19: Performance metrics of Neural Network models

Metrics	NNet Base Model	NNet Tuned Model
TDL	2.508	3.065
GINI	0.564	0.583
AUC	0.782	0.792
F1	91.152	96.434
Precision	98.485	97.949
Recall	84.835	94.966
Accuracy	83.891	93.131
Specificity	41.667	11.111
Time	25.228	39.743

The tuned model performs better than the base model across all key marketing metrics, with an increased targeting efficiency (TDL: 2.508 to 3.065) and discrimination capability (GINI: 0.564 to 0.583, AUC: 0.782 to 0.792). The higher F1-score (91.152 to 96.434) reflects enhanced recall (84.835% to 94.966%) while maintaining relatively strong precision (98.485% vs 97.949%), indicating superior identification of check-ins with minimal false positives. The accuracy also improved (83.891% to 93.131%) which further indicates the enhanced performance after tuning. Thus, the tuned NNet model was selected as the better fine-tuned model.

6 Performance Evaluation of Fine-Tuned Models

The comparative analysis of nine fine-tuned machine learning models indicates RF model has the best targeting efficiency (TDL = 3.901), meaning marketing campaigns reach nearly four times more engaged customers than random selection. Four models achieve identical strong performance (TDL = 3.343) are LG, KNN, SVM, and Boosting, while DT has the weakest targeting capability (TDL = 2.508). RF's exceptional lift performance, along with strong GINI (0.729) and high AUC (0.864), making it the most effective at identifying and ranking top customers, generating better campaign ROI for customer targeting. While Boosting model performs very well overall, especially with high GINI (0.749) and AUC (0.875), its advantage is rather in discrimination power, making it optimal for applications requiring precise customer probability estimation and risk assessment. Though NB model has strong discrimination (GINI = 0.762), low TDL and recall make it less suitable for lift-based marketing tasks.

Figure 6: Performance heatmap of nine fine-tuned models

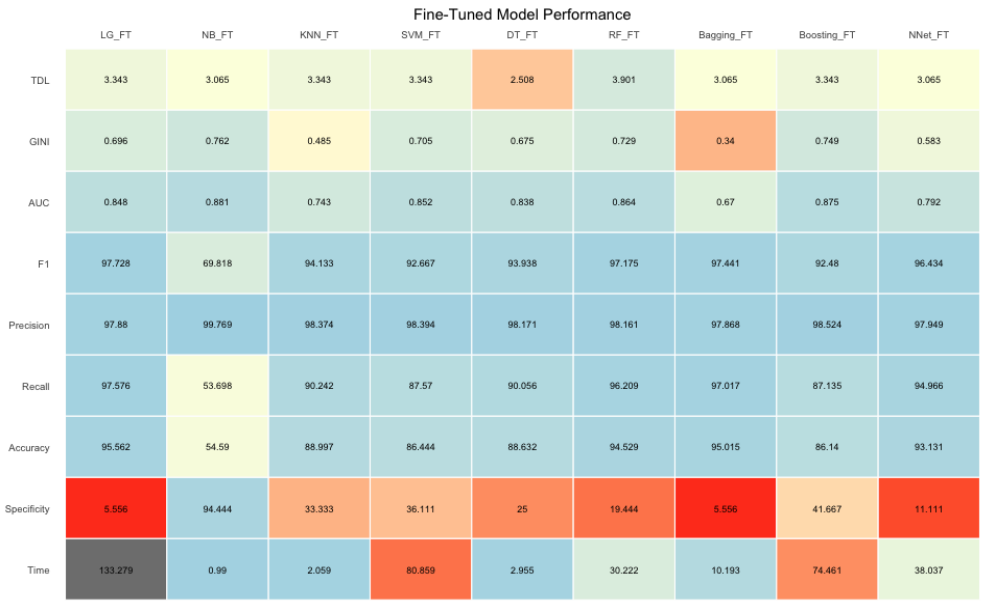


Figure 7: Lift curves and ROC curves of nine fine-tuned models

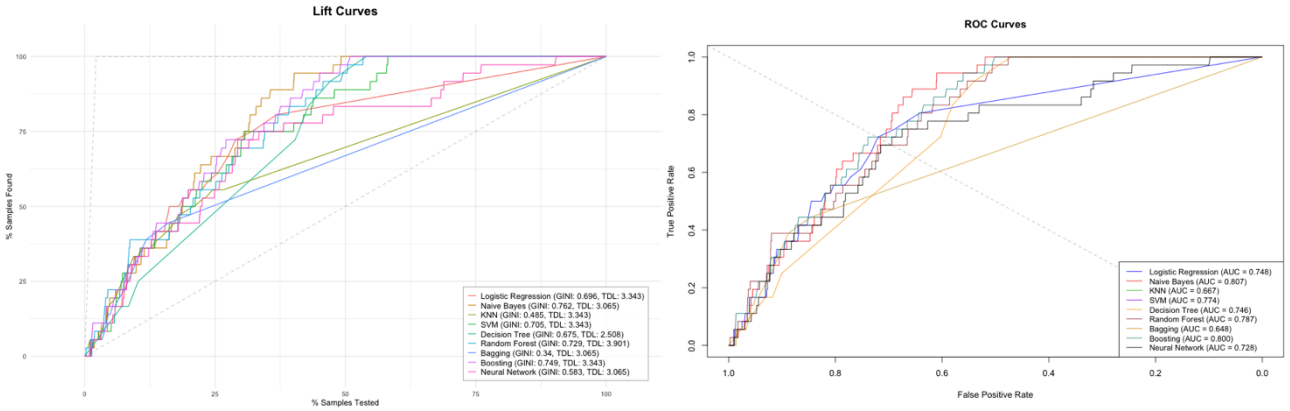
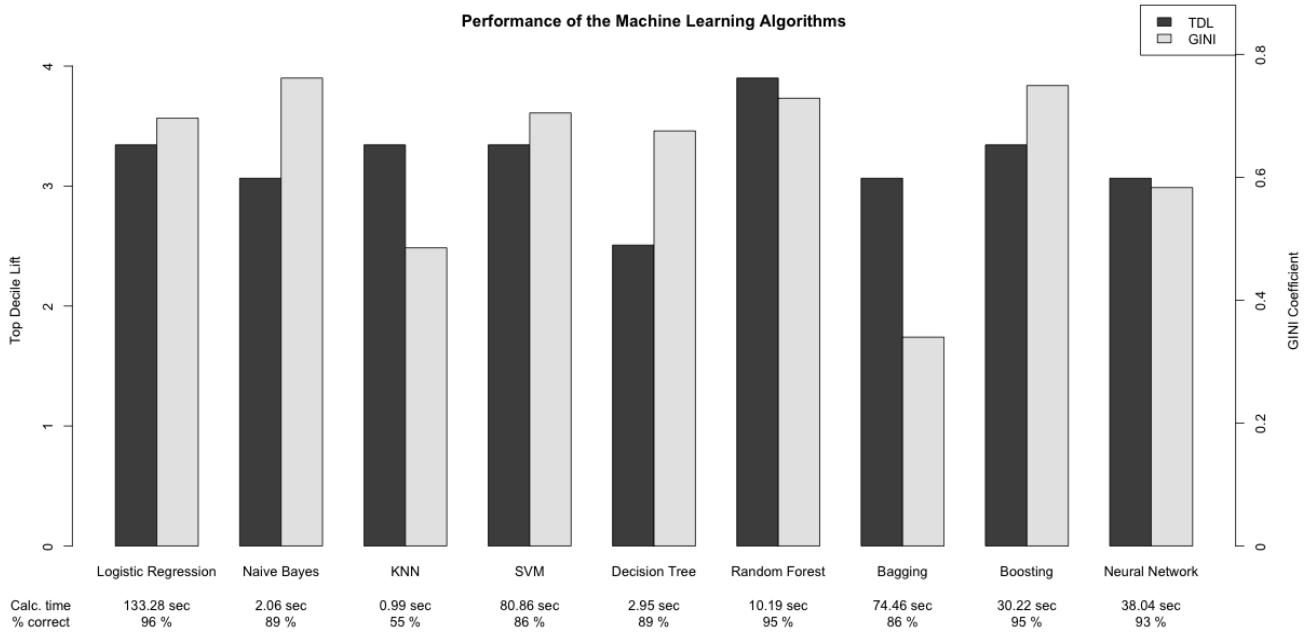


Figure 8: Performance graph of nine fine-tuned models



Overall, RF and Boosting are considered as the most optimal models for marketing analytics purposes, the selection depends on whether the marketing objective concentrates on targeting efficiency (RF model) or segment scoring capability (Boosting model).

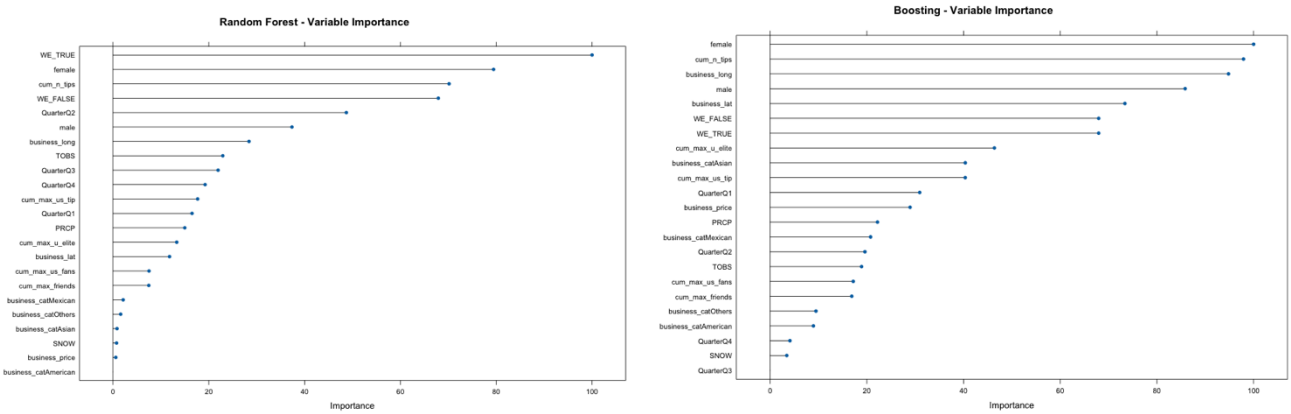
While RF is the best model for direct marketing campaigns and customer targeting, in which maximizing the concentration of top engaged customer segment is important for highest campaign efficiency and ROI in resource-constrained marketing initiatives. On the other hand, Boosting model is better for customer scoring systems and risk assessment applications where precise probability estimation across all customer segments is essential. Boosting provide better overall discrimination for portfolio management, customer lifetime value prediction, and engagement probability ranking.

7 Variable Importance

7.1 Causal Effect Interpretation

The variable importance analysis reveals which factors influence check-in behavior, which is the proxy for customer satisfaction according to the theoretical framework of this study.

Figure 9: Variable importance of Random Forest and Boosting models



Weekend, one of the temporal variables, is recognized as the most influential predictor across both models, top one and top tenth in RF and Boosting respectively. This temporal pattern reflects fundamental shifts in customer behavior, leisure time availability, and dining motivations that influence satisfaction through altered expectations and service experiences. Seasonal patterns also demonstrate moderate effects to the prediction outcome, with quarter 1 and 2 are among top factors.

Gender composition also represents the strong factor in both models, with female reviewer participation achieving the highest importance in Boosting and ranking second in RF. This pattern suggests that female-driven user-generated content serves as a stronger signal of restaurant quality and satisfaction drivers. Cumulative number of tips also shows consistent importance in two models, representing the influence where accumulated positive experiences may lead to increased check-in occurrences.

Location coordinates appear among top factors in both models, indicating that geographic positioning can influence customer satisfaction through accessibility, neighborhood characteristics, and competitive environment. Restaurant category shows model-dependent importance patterns, with Boosting has Asian as dominant category, while RF distributes importance across all four categories. This suggests satisfaction is moderately related to cuisine-specific factors.

For external variables, weather factors demonstrate modest but consistent effects across models, meaning weather conditions influence satisfaction through mood and mobility mechanisms. Temperature shows stronger effects than precipitation, suggesting customer satisfaction is more related to comfortable climate more than seasonal weather conditions.

7.2 Intervention Recommendation for Business Implications

Based on the causal interpretation of variable importance, the following intervention strategies are recommended to enhance customer satisfaction and check-in behavior.

7.2.1 Controllable Interventions

To maximize check-in behavior, which is the proxy for customer satisfaction, restaurants should prioritize interventions based on the importance identified. Weekend service optimization represents the most impactful strategy, requiring specialized weekend protocols including extended operating hours, enhanced staffing levels, special menu and promotion offerings. Besides being considered as one of the most influential factors in both predictive models, this intervention also has high impact and is controllable since restaurants can directly modify their weekend operations, staffing decisions, and service protocols.

Furthermore, female customer engagement programs can target the gender composition effect by implementing female-focused events, social media campaigns, review incentive programs and community activities that encourage female customer feedback and social sharing. As user-generated content variables are ranked among top factors, restaurant owners should also implement initiatives that encourage tip creation and review submission through loyalty and discount programs, which eventually enhance customer satisfaction and engagement perception through social proof mechanisms.

7.2.2 Strategic Interventions

To enhance check-in behavior through strategic long-term positioning, restaurants should implement interventions that require sustained commitment and broader operational changes. For instance, seasonal service may involve implementing quarterly specific, especially during summertime of quarter 2, operational adjustments including menu seasonality, service modifications, and promotional campaigns to achieve better check-in likelihood. Optimizing the presence in local area, as resonates with location-specific factors, through community engagement, partnerships with organizations within the neighborhood area, and location-tailored offerings also can enhance the restaurant's positioning within its geographic context.

7.2.3 Environmental Adaption Strategies

Lastly, restaurants should implement weather adaption strategies that deal with weather conditions, which moderately influence on check-in behavior and customer satisfaction. It should include in-store temperature-responsive operations, such as having air conditioners or heating and ventilation systems. These can influence check-in likelihood and customer satisfaction, particularly during extreme weather conditions. Moreover, long-term strategies for responding to weather conditions could also involve integrating a garden with trees into restaurant environment, which accordingly provide shading and temperature regulation. This approach is also considered as low-tech and cost-efficient for better environmental-friendly effects.

8 Conclusion and Outlook

This study successfully addresses the fundamental research question by using check-in as a proxy indicator, indicating that observable features such as weekend timing, female reviewer participation and user-generated content are recognized as the most influential factors to check-in behavior, subsequently customer satisfaction. Additionally, static restaurant attributes, temporal and external weather conditions also collectively influence customer satisfaction. Thus, intervention recommendations such as weekend service optimization, gender-focused engagement programs and weather-responsive strategies are suggested through casual interpretation for restaurant owners to enhance customer satisfaction.

Overall, Random Forest and Boosting models achieving the most optimal predictive performance (TDL = 3.901 and 3.343; GINI = 0.729 and 0.749) on the Yelp dataset with 6579 observations collective from restaurants in Saint Louis in two years. Thus, incorporating these algorithms in marketing planning may enhance campaigns to reach nearly four times more engaged customers than random selection, and achieve 70% improvement in customer segmentation.

Future research can expand geographic and temporal contexts across diverse restaurant types and markets, alongside with applying advanced causal machine learning algorithms like causal forests for heterogeneous treatment effect estimation for more comprehensive customer behavior measurement and intervention effectiveness. The approach of integrating predictive accuracy with causal interpretability represents a promising direction for marketing analytics applications where understanding causal mechanisms is essential for effective business decision-making and intervention design for restaurant owners.

References

Andaleeb, S. S., & Conway, C. (2006). Customer satisfaction in the restaurant industry: an examination of the transaction-specific model. *Journal of Services Marketing*, 20(1), 3–11.

Anderson, E. W., Fornell, C., & Mazvancheryl, S. K. (2004). Customer satisfaction and shareholder value. *Journal of Marketing*, 68(4), 172-185.

Bagozzi, R. P., & Yi, Y. (1988). On the evaluation of structural equation models. *Journal of the Academy of Marketing Science,* 16(1), 74-94.

Berry, M. J., & Linoff, G. S. (2004). *Data mining techniques: For marketing, sales, and customer relationship management* (2nd ed.). Wiley.

Bishop, C. M. (1995). *Neural networks for pattern recognition*. Oxford University Press.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2), 123-140.

Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32.

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and regression trees*. CRC Press.

Bühlmann, P., & Yu, B. (2002). Analyzing bagging. *The Annals of Statistics*, 30(4), 927-961.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, 20(3), 273-297.

Cover, T., & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 21-27.

Dwaikat, N. Y., Khalili, S. A., Hassis, S. M., & Mahmoud, H. S. (2019). Customer satisfaction impact on behavioral intentions: the case of pizza restaurants in Nablus City. *Journal of Quality Assurance in Hospitality & Tourism*, 20(6), 709–728.

Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 861-874.

Fornell, C., Johnson, M. D., Anderson, E. W., Cha, J., & Bryant, B. E. (1996). The American customer satisfaction index: Nature, purpose, and findings. *Journal of Marketing*, 60(4), 7-18.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1), 119-139.

Ha, J., & Jang, S. (2010). Effects of service quality and food quality: The moderating role of atmospherics in an ethnic restaurant segment. *International Journal of Hospitality Management*, 29(3), 520-529.

Hanley, J. A., & McNeil, B. J. (1982). The meaning and use of the area under a receiver operating characteristic (ROC) curve. *Radiology*, 143(1), 29-36.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

Hosmer, D. W., Lemeshow, S., & Sturdivant, R. X. (2013). *Applied logistic regression* (3rd ed.). Wiley.

Imbens, G. W., & Rubin, D. B. (2015). *Causal inference for statistics, social, and biomedical sciences*. Cambridge University Press.

Khademi, A., & Honavar, V.G. (2020). A Causal Lens for Peeking into Black Box Predictive Models: Predictive Model Interpretation via Causal Attribution. *ArXiv, abs/2008.00357*.

Kim, W. G., Ng, C. Y. N., & Kim, Y. S. (2009). Influence of institutional DINESERV on customer satisfaction, return intention, and word-of-mouth. *International Journal of Hospitality Management*, 28(1), 10-17.

Kumar, S., & Ravi, V. (2022). Application of causal inference to analytical customer relationship management in banking and insurance. *arXiv (Cornell University)*.

Louizos, C., Shalit, U., Mooij, J. M., Sontag, D., Zemel, R., & Welling, M. (2017). Causal Effect Inference with Deep Latent-Variable Models. *Advances in Neural Information Processing Systems*, 30.

Luca, M. (2011). Reviews, reputation, and revenue: The case of Yelp.com. *American Economic Journal: Applied Economics*, 3(3), 133-152.

Parida, P. K. (2021). Causality and its applications. In *Studies in computational intelligence* (pp. 205–221).

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1), 81-106.

Ripley, B. D. (1996). *Pattern recognition and neural networks*. Cambridge University Press.

Rubin, D. B. (1974). Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology*, 66(5), 688-701.

Ryu, K., & Han, H. (2019). Influence of the quality of food, service, and physical environment on customer satisfaction and behavioral intention in quick-casual restau-

rants: Moderating Role of Perceived Price. *Journal of Hospitality & Tourism Research*, 34(3), 310-329.

Scholkopf, B., & Smola, A. J. (2002). *Learning with kernels: Support vector machines, regularization, optimization, and beyond*. MIT Press.

Shalit, U., Johansson, F., & Sontag, D. (2017). Estimating Individual Treatment Effect: Generalization Bounds and Algorithms. *Proceedings of the 34th International Conference on Machine Learning (ICML)*.

Singh, S. S. K., Sinha, A. K., Pandey, T. N., & Acharya, B. M. (2023). A Machine Learning Approach to Compare Causal Inference Modelling Strategies in the Digital Advertising Industry. *2023 2nd International Conference on Ambient Intelligence in Health Care (ICAIHC)*, 1–7.

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427-437.

Stranjancevic, A. & Bulatovic, I. (2015). Customer satisfaction as an indicator of service quality in tourism and hospitality. *International Journal for Quality Research*. 9. 689-704.

Tarn, J. L. M. (1999). The effects of service quality, perceived value and customer satisfaction on behavioral intentions. *Journal of Hospitality & Leisure Marketing*, 6(4), 31–43.

Vargo, S. L., & Lusch, R. F. (2004). Evolving to a new dominant logic for marketing. *Journal of Marketing*, 68(1), 1-17.

Verhoef, P. C., Lemon, K. N., Parasuraman, A., Roggeveen, A., Tsiros, M., & Schlesinger, L. A. (2009). Customer experience creation: Determinants, dynamics and management strategies. *Journal of Retailing*, 85(1), 31-41.

Zhang, H. (2004). The optimality of naive Bayes. *AI 2004: Advances in Artificial Intelligence*, 3339, 562-567.

Appendix I: R Code

```r
#clear workspace
rm(list = ls())

# libraries
library(tidyverse)
library(dplyr)
library(doParallel)
library(caret)
library(smotefamily)
library(pROC)
library(ggplot2)
library(grid)
library(gridExtra)
library(stargazer)
library(reshape2)
library(fastDummies)
library(corrplot)
library(scales)
library(caret)
library(glmnet)
library(kernlab)
library(rpart)
library(ranger)
#library(ipred)
library(xgboost)
library(NeuralNetTools)
library(nnet)


# set working directory
setwd("/Users/tracie/Desktop/academic/goethe/2-
SoSe25/DSMA/[SUBMISSION]/TRAN_Ngoc-Khanh-Uyen_8518013 ")



# prepare tips dataset ------------------------------------------------------



# load the dataset
load("tipsbusiness_2020-01-01-2021-12-31_Korean_Saint Louis.RData")

# only select the non-business information,
# since the business information may include missing we need to deal with
tips_data = tips_data[,1:9]

# check for variables that are logical vector, and convert to integer
integer64s = sapply(tips_data,function(x){class(x)=="integer64"})
tips_data[,integer64s] = as.integer(tips_data[,integer64s])
```

```
# copy dataframe and extract names from cum_u_names column
DailyLevel_data = tips_data
cum_u_names = DailyLevel_data$cum_u_names

# count number of observations from cum_u_names column
Nobs=length(cum_u_names)

# replacing the cum_u_names with the count of reviewers' genders
if(0){
  #install.packages("gender")
  library(gender)
  #remotes::install_github("ropensci/gender-data-pkg")
  library(genderdata)
  # best is to do gender extractin in parallel
  library(doParallel)


  gendersplit=function(x){
    a=max.col(gender(unlist(strsplit(x,",")))[,2:3])
    return(c(male=sum(a==1,na.rm=T),female=sum(a==2,na.rm=T)))
  }

  cl=makeCluster(detectCores()/2+2)
  registerDoParallel(cl)
  nameslist=NULL
  for(k in 1:20){
    whichrun=floor(Nobs/20*(k-1)+1):floor(Nobs/20*k)
    a=foreach(i=whichrun,.packages=c("gender"),.noexport =
c("DailyLevel_data"),.combine=rbind) %dopar%
      {gendersplit(cum_u_names[i])}
    rownames(a)=NULL
    nameslist=rbind(nameslist,a)
    print(k)
  }
  stopImplicitCluster()
  stopCluster(cl)
  save(file="nameslist_rev.RData",list="nameslist")
}else{
  load("nameslist_rev.RData")
}

# add 2 gender count columns into dataset, and remove cum_u_names column
DailyLevel_data = cbind(DailyLevel_data,nameslist)
DailyLevel_data$cum_u_names = NULL

# store the date in date column as correct format, and remove the old name
DailyLevel_data$date <- as.Date(DailyLevel_data$date_tip)
DailyLevel_data$date_tip <- NULL



# merge with business data ------------------------------------------------------
```

```
# load business data
load("business_Korean_Saint Louis.RData")

# check for variables that are logical vector, and convert to integer
integer64s = sapply(output_business,function(x){class(x)=="integer64"})
output_business[,integer64s] = as.integer(output_business[,integer64s])
business_data = output_business

# dealing with the missings
business_data$n_photo[is.na(business_data$n_photo)] = 0

business_data1 <- subset(business_data,select = -c(business_id))
# removed this because MICE does not like imputing factors with more than 50
levels

library(mice)

#inspect pattern of missings
md.business=md.pattern(business_data1)

if(md.business[length(md.business)]!=0){ # do the imputation only if there are
missings
  # Below, the predictormatrix is specified
  # It is a square matrix of size ncol(data) containing 0/1 data specifying the
set of predictors to be used for each target column
  # Rows correspond to target variables (i.e. variables to be imputed), in the
sequence as they appear in data
  # A value of '1' means that the column variable is used as a predictor for the
target variable (in the rows)
  # The diagonal of predictorMatrix must be zero
  predictorMatrix <- matrix(0,nrow = ncol(business_data1), ncol =
ncol(business_data1)) # Make a matrix of zeros
  colnames(predictorMatrix)=colnames(business_data1)
  row.names(predictorMatrix)=colnames(business_data1)
  predictorMatrix[c("business_price"),] <- 1 #variables "business_price" can be
explained by all other variables
  diag(predictorMatrix) <- 0 #diagonal must be zero

  #impute data
  business_data1_data_imputed <- mice(business_data1, predictorMatrix =
predictorMatrix, m=5, maxit = 50, seed = 500)

  summary(business_data1_data_imputed)

  #get one of the complete data sets ( 2nd out of 5)
  business_data_complete_data <- complete(business_data1_data_imputed,2)

  # bring back the business_id
```

```
business_data_complete_data=cbind(business_id=business_data$business_id,business_
data_complete_data)

}else{
  business_data_complete_data = business_data
}

# merge 2 dataset DailyLevel_data & business_data_complete_data using business id
column
DailyLevel_data = DailyLevel_data%>%
  inner_join(business_data_complete_data,by="business_id")


# make factors out of chr variables
for(j in 1:ncol(DailyLevel_data)){
  if(typeof(DailyLevel_data[,j])=="character")
    DailyLevel_data[,j]=as.factor(DailyLevel_data[,j])
}

# limit the number of categories to Asian, American, Mexican and Others
cat_s=as.character(DailyLevel_data$business_cat)
new_cat_s=c("Others","Asian", "American", "Mexican")

changed=0
for(k in new_cat_s[-1]){
  cat_s[grepl(k,cat_s)]=k
  changed=changed+grepl(k,cat_s)
}
cat_s[changed==0]="Others"
DailyLevel_data$business_cat=as.factor(cat_s)

# n_photos==NA and cum_max_u_elite==NA are actually zeros, therefore, replace
them with 0 before imputing
DailyLevel_data$cum_max_u_elite[is.na(DailyLevel_data$cum_max_u_elite)]=0

# check descriptives of the data
Hmisc::describe(DailyLevel_data)


#the complete dataset can be used to estimate predictive models
write.csv(DailyLevel_data, file="DailyLevel_data_tip_Imputed.csv")



# deploy of the functions used for the analysis ------------------------------


extractweather=function(dataset,mindate=min(dataset$date),maxdate=max(dataset$dat
e),

latrange=range(dataset$business_lat),longrange=range(dataset$business_long),
```

```
                          resol=.5,getdata=FALSE,

wear=ifelse("weatherPRCPSNWDSNOWTMAXTMINTOBS.RData"%in%list.files(),"available","
navailable"),
                          cl=NULL){
  # queries weather data from ncdc.noaa.gov, in a grid format from mindate until
maxdate
  # if not specified, takes the min and maxdate from dataset
  # the geographical range is latitudinal in latrange and longitudinal in long-
range
  # the resolution of the grids is determined by resol
  # the data is stored in weatherPRCPSNWDSNOWTMAXTMINTOBS.RData ,
  # if already existing, the weather data is no more extracted

  wdatacond=wear=="navailable"
  if(getdata | wdatacond){
    if(is.null(cl)){
      require("doParallel")

      cl <- makeCluster(detectCores(),outfile="log1.txt")
      registerDoParallel(cl)
    }


    # read the station names

stations=read.delim(url("https://www1.ncdc.noaa.gov/pub/data/ghcn/daily/ghcnd-
stations.txt"),header = F,quote="",sep="")[,1:3]
    colnames(stations)=c("Station","lat","long")
    stations=stations[strtrim(stations$Station,2)=="US",]
    stations$lat=as.numeric(stations$lat)
    stations$long=as.numeric(stations$long)
    stations=stations[!is.na(stations$lat)|!is.na(stations$long),]

    # mindate=min(dataset$date)#"2016-05-01"#
    # maxdate=max(dataset$date)#"2016-05-02"#
    # latrange=range(dataset$business_lat)
    # longrange=range(dataset$business_long)

    latseq=c(seq(latrange[1],latrange[2],by=resol),latrange[2])
    longseq=c(seq(longrange[1],longrange[2],by=resol),longrange[2])

    wear=NULL
    k=0
    torunlist=NULL
    for(lat in 1:(length(latseq)-1)){#(length(latseq)-1)
      for(lon in 1:(length(longseq)-1)){
        k=k+1
        torunlist=rbind(torunlist,c(lat,lon))
      }
    }
```

```
wear=foreach(i=1:k,.noexport=ls(),.export=c("latseq","longseq","stations","torunl
ist","mindate","maxdate"))%dopar%
      {
        # find the station(s) within the boxes
        lat=torunlist[i,1]
        lon=torunlist[i,2]
        rangelat=c(latseq[lat+1],latseq[lat])
        rangelong=c(longseq[lon],longseq[lon+1])

indx=(stations$lat>rangelat[2])&(stations$lat<rangelat[1])&(stations$long>rangelo
ng[1])&(stations$long<rangelong[2])
        stations_temp=stations[indx,]
        stations_t=paste(stations_temp$Station,collapse=",")
        temp=paste0("dataset=daily-
summaries&dataTypes=PRCP,SNWD,SNOW,TMAX,TMIN,TOBS",

"&stations=",stations_t,"&startDate=",mindate,"","&endDate=",maxdate)#,

#"","&boundingBox=",paste(latseq[lat+1],longseq[lon],latseq[lat],longseq[lon+1],s
ep=","))##90,-180,-90,180
        valid_url <- TRUE

a=tryCatch(read.csv(url(paste0("https://www.ncei.noaa.gov/access/services/data/v1
?",temp))),error=function(e) {valid_url<<-FALSE})
        toreturn=NULL
        if(valid_url)

toreturn=list(range=cbind(rangelat,rangelong),data=read.csv(url(paste0("https://w
ww.ncei.noaa.gov/access/services/data/v1?",temp))))
        print(c(lat,lon,valid_url))
        return(toreturn)
        #print(c(lat,lon,valid_url))
      }

    #on.exit(stopCluster(cl))
    stopCluster(cl)
    save(file="weatherPRCPSNWDSNOWTMAXTMINTOBS.RData",list=c("wear"))
  }else{
    if(wear=="available"){
      load("weatherPRCPSNWDSNOWTMAXTMINTOBS.RData")
    }
  }
  return(wear)
}


weardailyavg=function(wear){
  # this function converts the extracted weather data into daily level data.
  if("weather_data.RData"%in%list.files()){
    load(file="weather_data.RData")
  }else{
    require("doParallel")
```

```
    cl <- makeCluster(detectCores())
    registerDoParallel(cl)
    clusterCall(cl,function(x) {library(dplyr)})
    wear_avg=NULL
    k=0
    wear_avg=foreach(i=1:length(wear),.noexport=ls(),.export=c("wear"),.packages
= c("dplyr"))%dopar%
      {
        if(is.null(wear[[i]])){
          temp=NULL
        }else{
          temp=wear[[i]]$data %>%
            group_by(DATE) %>%
            summarize(PRCP=mean(PRCP,na.rm = T),SNOW=mean(SNOW,na.rm =
T),SNWD=mean(SNWD,na.rm = T),
                      TMAX=mean(TMAX,na.rm = T),TMIN=mean(TMIN,na.rm =
T),TOBS=mean(TOBS,na.rm = T))
          temp=list(range=wear[[i]]$range,data=temp)}
        return(temp)

      }
    stopCluster(cl)
    weather=NULL
    k=0
    for(i in 1:length(wear_avg)){
      if(is.null(wear[[i]]))
        next
      k=k+1
      weather[[k]]=wear_avg[[i]]
      weather[[k]]$data$DATE=as.Date(weather[[k]]$data$DATE)
    }
    save(file="weather_data.RData",list=c("weather"))
  }
  return(weather)
}


makeLiftPlot <- function(Prediction, Evaluate, ModelName){
  # plots the liftplot, and computes the GINI coefficient.
  iPredictionsSorted <- sort(Prediction,index.return=T,decreasing=T)[2]$ix
#extract the index order according to predicted retention
  CustomersSorted <- Evaluate$ch_in_string[iPredictionsSorted] #sort the true
behavior of customers according to predictions
  SumChurnReal<- sum(Evaluate$ch_in_string == "ch_in") #total number of real
churners in the evaluation set
  CustomerCumulative=seq(nrow(Evaluate))/nrow(Evaluate) #cumulative fraction of
customers
  ChurnCumulative=apply(matrix(CustomersSorted=="ch_in"),2,cumsum)/SumChurnReal
#cumulative fraction of churners
  ProbTD =
sum(CustomersSorted[1:floor(nrow(Evaluate)*.1)]=="ch_in")/floor(nrow(Evaluate)*.1
) #probability of churn in 1st decile
```

```
  ProbOverall = SumChurnReal / nrow(Evaluate) #overall churn probability
  TDL = ProbTD / ProbOverall
  GINI = sum((ChurnCumulative-CustomerCumulative)/(t(matrix(1,1,nrow(Evaluate))-
CustomerCumulative)),na.rm=T)/nrow(Evaluate)
  plot(CustomerCumulative,ChurnCumulative,type="l",main=paste("Lift Curve of",
ModelName),xlab="Cumulative fraction of check-ins (sorted by predicted check-in
probability)",ylab="Cumulative fraction of check-ins")
  lines(c(0,1),c(0,1),col="blue",type="l",pch=22, lty=2)
  legend(.66,.2,c("According to model","Random selection"),cex=0.8,
col=c("black","blue"), lty=1:2)
  text(0.15,1,paste("TDL = ",round(TDL,2), "; GINI = ", round(GINI,2) ))
  return(data.frame(TDL,GINI))
}



# extract the weather data, and add it to the yelp data -------------------------



if(0){
  #load data
  yelp_data <- read.csv("DailyLevel_data_tip_Imputed.csv",header=TRUE,skipNul =
T) #read csv file
  yelp_data$date <- as.Date(yelp_data$date)
  yelp_data$X=NULL

  # read the temperature data

  rangeX=.01
  latrange=range(yelp_data$business_lat)*c(1-rangeX,1+rangeX)
  longrange=range(yelp_data$business_long)*c(1+rangeX,1-rangeX)

  cl <- makeCluster(detectCores(),outfile="log1.txt")
  registerDoParallel(cl)

  wear=extractweather(yelp_data,latrange=latrange,longrange=longrange,resol=.25,
cl=cl)

  nocl=FALSE
  tryCatch(stopCluster(cl),error=function(e) {nocl<<-TRUE})
  if(!nocl){
    stopCluster(cl)
  }

  # take the averages across stations for each coordinate
  weather=weardailyavg(wear)


  dates=sort(unique(yelp_data$date))

weatherstations=as.data.frame(t(sapply(weather,function(x){colMeans(x$range)})))
```

```
  # adding weather data to yelp_data
  if(1){
    stations_by=t(apply(yelp_data[,c("business_lat","business_long")],1,
                        function(x){a=sort((x[1]-weatherstations$rangelat)^2+
                                             (x[2]-
weatherstations$rangelong)^2,index.return=T)
                        return(a$ix[1:50])})) # finding the 50 closest stations

    # add for example, temperature forecasts to the weather data
    for(i in 1:length(weather)){
      if(nrow(weather[[i]]$data)==0)
        next
      store_weather=weather[[i]]$data
      store_weather$TOBS_1=c(store_weather$TOBS[2:nrow(store_weather)],NA)
      store_weather$TOBS_2=c(store_weather$TOBS[3:nrow(store_weather)],NA,NA)
      store_weather$TOBS_3=c(store_weather$TOBS[4:nrow(store_weather)],NA,NA,NA)

store_weather$TOBS_4=c(store_weather$TOBS[5:nrow(store_weather)],NA,NA,NA,NA)
      weather[[i]]$data=store_weather
    }
    weatherinf=colnames(store_weather)[-1]

    yelp_data_weather=NULL
    for(i in 1:length(weather)){
      k=1 # start with the closest station
      stores_in=stations_by[,k]==i
      if(sum(stores_in)==0)
        next
      store_weather=weather[[i]]$data

      temp=yelp_data[stores_in,]
      temp=merge(temp,store_weather,by.x="date",by.y="DATE",all.x=T)
      yelp_data_weather=rbind(yelp_data_weather,temp)
      print(i)
    }

    # now deal with the missings, by going to the next possible station
    temp_indx=is.na(yelp_data_weather[,"TOBS"])|is.na(yelp_data_weather[,"PRCP"])
    k_changed=NULL
    for(i in which(temp_indx)){
      temp_date=yelp_data_weather[i,]$date
      for(k in 2:ncol(stations_by)){
        temp=weather[[stations_by[i,k]]]$data

if(!is.na(as.numeric(temp[temp$DATE==temp_date,"TOBS"]))&!is.na(as.numeric(temp[t
emp$DATE==temp_date,"PRCP"])))
          break
      }
      k_changed=c(k_changed,k)

      yelp_data_weather[i,weatherinf]=temp[temp$DATE==temp_date,-1]
      #print(i)
```

```
    }

    # add weekends and quarters
    temp=weekdays(yelp_data_weather$date,abbreviate = T)
    yelp_data_weather$WE=temp=="Sat"|temp=="Sun"

    yelp_data_weather$Quarter=as.factor(quarters(yelp_data_weather$date))

    #save(file="yelp_data_weather.RData",list=c("yelp_data_weather"))
    write.csv(yelp_data_weather,file="yelp_data_tip_weather.csv")

  }

}



# adjusting the yelp-data + weather data ----------------------------------------



# load the dataset
yelp_data_weather = read.csv(file="yelp_data_tip_weather.csv")

# adjustments to the imported data

yelp_data = yelp_data_weather

yelp_data$date = as.Date(yelp_data$date)
yelp_data$ch_in_string[yelp_data$ch_in>=1]="ch_in"
yelp_data$ch_in_string[yelp_data$ch_in==0]="Noch_in"
yelp_data$ch_in_string <- as.factor(yelp_data$ch_in_string)

# since the performance evaluations are mainly made to check for the minority
class, in our case "ch_in"
# therefore, set "noch_in" set reference level
yelp_data$ch_in_string <- relevel(yelp_data$ch_in_string,ref="Noch_in")

# convert variables to categorical data
yelp_data$business_park=as.factor(yelp_data$business_park)
yelp_data$business_open=as.factor(yelp_data$business_open)
yelp_data$business_cat=as.factor(yelp_data$business_cat)
yelp_data$WE=as.factor(yelp_data$WE)
yelp_data$Quarter=as.factor(yelp_data$Quarter)



# statistics & correlations of dataset------------------------------------------



numeric_vars <- sapply(yelp_data, is.numeric)
numeric_vars["X"] <- FALSE # remove variable X which is row number from csv file
```

```
yelp_numeric <- yelp_data[, numeric_vars]

stargazer(yelp_numeric, type = "text", digits = 3, summary.stat = c("n", "mean",
"sd", "min", "median", "max"))

cor_matrix <- cor(yelp_numeric, use = "pairwise.complete.obs")

corrplot(cor_matrix,
         method = "color",
         type = "upper",
         tl.cex = 0.8,
         tl.col = "black",
         number.cex = 0.7)



# split business_cat, business_open, WE, Quarter variables-----------------------



business_cat_dummies <- model.matrix(~ business_cat - 1, data = yelp_data)
yelp_data <- cbind(yelp_data, business_cat_dummies)

yelp_data$business_open <- as.logical(yelp_data$business_open)
yelp_data$business_open_TRUE <- ifelse(yelp_data$business_open == TRUE, 1, 0)
yelp_data$business_open_FALSE <- ifelse(yelp_data$business_open == TRUE, 1, 0)

yelp_data$WE <- as.logical(yelp_data$WE)
yelp_data$WE_TRUE <- ifelse(yelp_data$WE == TRUE, 1, 0)
yelp_data$WE_FALSE <- ifelse(yelp_data$WE == FALSE, 1, 0)


yelp_data$Quarter <- as.factor(yelp_data$Quarter)
quarter_dummies <- model.matrix(~ Quarter - 1, data = yelp_data)
yelp_data <- cbind(yelp_data, quarter_dummies)

yelp_data$WE <- NULL
yelp_data$Quarter <- NULL
yelp_data$business_cat <- NULL
yelp_data$business_open <- NULL



# variable grouping and explanation----------------------------------------------



# 1) Static variables (physical/business attributes)

business_cat_cols <- grep("^business_cat", colnames(yelp_data), value = TRUE)
static_vars <- c("ch_in", "business_id", "business_open_TRUE",
"business_open_FALSE",
                 "business_price",
```

```
                "business_lat", "business_long", "business park",
                business_cat_cols)

static_meaning <- c(
  "Check-in indicator",
  "Unique business identifier",
  "Business is open",
  "Business is not open",
  "Business price category",
  "Latitude coordinate of business",
  "Longitude coordinate of business",
  "Business park",
  rep("Business category", length(business_cat_cols))
)

static_table <- data.frame(Variable = static_vars, Meaning = static_meaning)

print("Static variables (physical/business attributes)")
print(static_table)

# 2) Dynamic variables (user/activity data)
dynamic_vars <- c("ch_in", "cum_n_tips", "cum_max_friends", "cum_max_u_elite",
"cum_max_us_fans",
                  "cum_max_us_tip", "male", "female", "n_photo")

dynamic_meaning <- c(
  "Check-in indicator",
  "Cumulative number of tips",
  "Cumulative max friends",
  "Cumulative max elite users",
  "Cumulative max fans",
  "Cumulative max tips by users",
  "Count of male users involved",
  "Count of female users involved",
  "Number of photos"
)

dynamic_table <- data.frame(Variable = dynamic_vars, Meaning = dynamic_meaning)

print("Dynamic variables (user-generated data)")
print(dynamic_table)

# 3) Temporal variables (time-related)
temporal_vars <- c("ch_in", "WE_TRUE", "WE_FALSE", grep("^Quarter",
colnames(yelp_data), value=TRUE))

temporal_meaning <- c(
  "Check-in indicator",
  "Day is weekend",
  "Day is not weekend",
  paste(gsub("Quarter", "Quarter ", grep("^Quarter", colnames(yelp_data),
value=TRUE)))
```

```
)

temporal_table <- data.frame(Variable = temporal_vars, Meaning =
temporal_meaning)

print("Temporal variables (time-related)")
print(temporal_table)

# 4) External variables (weather data)
external_vars <- c("ch_in", "PRCP", "SNOW", "SNWD", "TMAX", "TMIN", "TOBS",
"TOBS_1", "TOBS_2", "TOBS_3", "TOBS_4")

external_meaning <- c(
  "Check-in indicator",
  "Precipitation amount",
  "Snowfall amount",
  "Snow depth",
  "Maximum temperature",
  "Minimum temperature",
  "Observed temperature",
  "Observed temperature at hour 1",
  "Observed temperature at hour 2",
  "Observed temperature at hour 3",
  "Observed temperature at hour 4"
)

external_table <- data.frame(Variable = external_vars, Meaning =
external_meaning)

print("External variables (weather data)")
print(external_table)



# statistics & correlations of data groups-------------------------------------



get_numeric_corr <- function(vars, data){
  vars_exist <- vars[vars %in% colnames(data)] # select only variables that exist
and are numeric
  numeric_vars <- vars_exist[sapply(data[, vars_exist], is.numeric)]
  if(length(numeric_vars) < 2){
    message("Not enough numeric variables for correlation in category.")
    return(NULL)
  }
  cor_mat <- cor(data[, numeric_vars], use = "complete.obs")
  return(list(cor_mat = cor_mat, vars = numeric_vars))
}

plot_corr <- function(cor_mat, title){
  corrplot(cor_mat, method = "color", type = "upper", tl.cex = 0.8, tl.col =
"black",
```

```
           number.cex = 0.7, main = title)
}

# static variables
static_corr <- get_numeric_corr(static_vars, yelp_data)
if(!is.null(static_corr)) plot_corr(static_corr$cor_mat, title = "")
stargazer(yelp_data[, static_corr$vars], type = "text",
          digits = 3, summary.stat = c("n", "mean", "sd", "min", "median",
"max"))

# dynamic variables
dynamic_corr <- get_numeric_corr(dynamic_vars, yelp_data)
if(!is.null(dynamic_corr)) plot_corr(dynamic_corr$cor_mat, title = "")
stargazer(yelp_data[, dynamic_corr$vars], type = "text",
          digits = 3, summary.stat = c("n", "mean", "sd", "min", "median",
"max"))

# temporal variables
temporal_corr <- get_numeric_corr(temporal_vars, yelp_data)
if(!is.null(temporal_corr)) plot_corr(temporal_corr$cor_mat, title = "")
stargazer(yelp_data[, temporal_corr$vars], type = "text",
          digits = 3, summary.stat = c("n", "mean", "sd", "min", "median",
"max"))

# external variables
external_corr <- get_numeric_corr(external_vars, yelp_data)
if(!is.null(external_corr)) plot_corr(external_corr$cor_mat, title = "")
stargazer(yelp_data[, external_corr$vars], type = "text",
          digits = 3, summary.stat = c("n", "mean", "sd", "min", "median",
"max"))



# predictive  models ------------------------------------------------------------



# drop columns due to low predictive power and redundant: X, date, business_id,
n_photo
#                                                    business_open_TRUE,
business_open_FALSE,
#                                                    business_park, SNWD,
TMAX, TMIN,
#                                                    TOBS_1, TOBS_2, TOBS_3,
TOBS_4

# split the dataset randomly
set.seed(66)

# make a copy of original data
yelp_data_na=yelp_data

# list of variables in the model
```

```
varsin=c("ch_in_string","ch_in","WE_TRUE","WE_FALSE",
         "QuarterQ1", "QuarterQ2", "QuarterQ3", "QuarterQ4",
         "business_price",
         "business_catAmerican", "business_catAsian", "business_catMexican",
"business_catOthers",
         "TOBS","PRCP", "SNOW",
         "business_lat", "business_long",
         "female","male","cum_n_tips","cum_max_friends",
         "cum_max_u_elite","cum_max_us_fans","cum_max_us_tip")


# subset data to the selected columns
yelp_data=subset(yelp_data,select=varsin)

# check size of dataset
datasetsize=nrow(yelp_data)/1

# randomly rearrange the dataset
x <- yelp_data[sample(1:nrow(yelp_data), datasetsize, replace = F),]

# split dataset into train and evaluate sets
# split ratio: 72:25
x.train <- x[1:floor(nrow(x)*.75), ]
x.evaluate <- x[(floor(nrow(x)*.75)+1):nrow(x), ]

# set up model formulas
# predict ch_in_string (factor version: "ch_in" or "Noch_in") using all other
variables except ch_in_string and ch_in
BaseFormula <- as.formula(paste0("ch_in_string~",paste(varsin[-c(1,2)],collapse =
"+")))
# predict ch_in (numeric/binary) using the same predictors
BaseFormula1 <- as.formula(paste0("ch_in~",paste(varsin[-c(1,2)],collapse =
"+")))


# create dummies (required for SMOTE)
x.traindum=cbind(x.train[,c("ch_in","ch_in_string")],predict(dummyVars(BaseFormul
a1,data=x.train),newdata = x.train))
x.evaluatedum=cbind(x.evaluate[,c("ch_in","ch_in_string")],predict(dummyVars(Base
Formula1,data=x.evaluate),newdata = x.evaluate))

# class imbalance check
temp=table(x.train[,"ch_in_string"])
print(temp)

# if yes, do random over-sampling:
if(0){

oversampled=x.train[x.train$ch_in_string==names(temp)[sort.int(temp,index.return=
T,decreasing = T)$ix[1]],]
  minclass=names(temp)[sort.int(temp,index.return=T)$ix[1]]
  for(m in 1:(length(temp)-1)){
    minchclass=names(temp)[sort.int(temp,index.return=T)$ix[m]]
```

```
    minclassdat=x.train[x.train$ch_in_string==minchclass,]
    minclassdat=minclassdat[sample(1:nrow(minclassdat), sort(temp,decreasing =
T)[1] , replace = T),]
    oversampled=rbind(oversampled,minclassdat)
  }
  x.train=oversampled
}

# or do SMOTE:
if(1){
  x.traindum_smote=SMOTE(x.traindum[,-c(1,2)],x.traindum[,2])$data
  names(x.traindum_smote)[ncol(x.traindum_smote)]="ch_in_string"
  x.traindum_smote$ch_in=ifelse(x.traindum_smote$ch_in_string=="ch_in",1,0)
  x.traindum_smote$ch_in_string=as.factor(x.traindum_smote$ch_in_string)
  x.traindum=x.traindum_smote
  rm(x.traindum_smote)
}

sum(is.na(x.traindum[,-c(1,2)]))
sum(is.na(x.traindum[,2]))
colSums(is.na(x.traindum[,-c(1,2)]))

# check for class distribution again
temp=table(x.traindum[,"ch_in_string"])
print(temp)

# there is no class imbalance now


# data for Heuristic machine learning methods

# normalize data (for ML techniques, except logistic regression)

x.trainnorm=predict(preProcess(x.traindum, method = "range"), newdata=x.traindum)
x.evaluatenorm=predict(preProcess(x.evaluatedum, method = "range"),
newdata=x.evaluatedum)

# adjust Baseformula to the  version of the data
varsin_dum=varsin[1:2]
for(i in 3:length(varsin)){
  if(!is.null(levels(x[,varsin[i]]))){
    for(j in 2:nlevels(x[,varsin[i]])){ # first level will be considered as the
base-level
      varsin_dum=c(varsin_dum,paste(varsin[i],levels(x[,varsin[i]])[j],sep="."))
    }
  }else{
    varsin_dum=c(varsin_dum,varsin[i])
  }
}

# redo the releveling:
x.traindum$ch_in_string=relevel(x.traindum$ch_in_string,ref="Noch_in")
x.evaluatedum$ch_in_string=relevel(x.evaluatedum$ch_in_string,ref="Noch_in")
```

```
x.trainnorm$ch_in_string=relevel(x.trainnorm$ch_in_string,ref="Noch_in")
x.evaluatenorm$ch_in_string=relevel(x.evaluatenorm$ch_in_string,ref="Noch_in")


BaseFormula_dum <- as.formula(paste0("ch_in_string~",paste(varsin_dum[-
c(1,2)],collapse = "+")))
BaseFormula1_dum <- as.formula(paste0("ch_in~",paste(varsin_dum[-c(1,2)],collapse
= "+")))

# set threshold probability, usually .5,
# but better is to set it to the portion of 1's, rather than the default 0.5,
which is especially useful for imbalanced datasets
probthres=mean(x.traindum$ch_in)



# 10-fold cross validation settings ------------------------------------------



ctrl <- trainControl(
  method = "cv",
  number = 10,
  search = "grid",
  classProbs = TRUE,
  savePredictions = "final",
  allowParallel = FALSE
)



# model training and hyperparameter tunning ----------------------------------



# function for evaluation metrics
get_metrics <- function(pred, pred_class, reference, tdl, gini, time_elapsed =
NA) {
  cm <- confusionMatrix(factor(pred_class, levels = c("Noch_in", "ch_in")),
                        factor(reference, levels = c("Noch_in", "ch_in")))
  auc <- (gini + 1) / 2
  #rocobj <- tryCatch({
  #  roc(actual, probs, levels = c("Noch_in", "ch_in"), direction = "<")
  #}, error=function(e) NA)
  #auc <- if (is.na(rocobj)[1]) NA else as.numeric(auc(rocobj))
  #auc <- tryCatch({
  #  rocobj <- pROC::roc(reference == "ch_in", pred)
  #  as.numeric(pROC::auc(rocobj))
  #}, error = function(e) NA)
  precision <- cm$byClass["Precision"]
  recall <- cm$byClass["Recall"]
  specificity <- cm$byClass["Specificity"]
  f1 <- cm$byClass["F1"]
```

```
  accuracy <- as.numeric(cm$overall["Accuracy"])
  metrics <- c(
    TDL = tdl,
    GINI = gini,
    AUC = auc,
    F1 = f1 * 100,
    Precision = precision * 100,
    Recall = recall * 100,
    Accuracy = accuracy * 100,
    Specificity = specificity * 100,
    Time = time_elapsed
  )
  names(metrics) <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall",
"Accuracy", "Specificity", "Time")
  return(metrics)
}



######### 1) LOGISTIC REGRESSION ----------------------------------------------

### BASE MODEL

ptm_base <- proc.time()

set.seed(123)
x.modelLogit_base <- train(BaseFormula_dum, data = x.traindum,
                           family = "binomial", trControl = ctrl)
summary(x.modelLogit_base)

base_time <- proc.time() - ptm_base

x.evaluate$predictionlogit_base <- predict(x.modelLogit_base,
newdata=x.evaluatedum, type = "prob")[, "ch_in"]
x.evaluate$predictionlogitclass_base <- ifelse(x.evaluate$predictionlogit_base >
probthres, "ch_in", "Noch_in")
x.evaluate$correctlogit_base <- x.evaluate$predictionlogitclass_base ==
x.evaluate$ch_in_string

LogitOutput_base <- makeLiftPlot(x.evaluate$predictionlogit_base, x.evaluate,
"Logistic Regression (Base Model)")
LogitOutput_base$PercCorrect <- mean(x.evaluate$correctlogit_base)*100
Logitconfmatrix_base <- table(x.evaluate$predictionlogitclass_base,
x.evaluate$ch_in_string)
LogitOutput_base$TimeElapsed <- base_time[3]

### HYPERPARAMETER TUNING

tune_grid <- expand.grid(
  alpha = 2,
  lambda = 0.05
)
```

```
ptm_tune <- proc.time()

set.seed(123)
x.modelLogit_tuned <- train(
  BaseFormula_dum, data = x.traindum, method = "glmnet",
  family = "binomial",
  trControl = ctrl,
  tuneGrid = tune_grid
)

tuned_time <- proc.time() - ptm_tune

print(x.modelLogit_tuned$bestTune)

x.evaluate$predictionlogit_tuned <- predict(x.modelLogit_tuned,
newdata=x.evaluatedum, type = "prob")[, "ch_in"]
x.evaluate$predictionlogitclass_tuned <- ifelse(x.evaluate$predictionlogit_tuned
> probthres, "ch_in", "Noch_in")
x.evaluate$correctlogit_tuned <- x.evaluate$predictionlogitclass_tuned ==
x.evaluate$ch_in_string

LogitOutput_tuned <- makeLiftPlot(x.evaluate$predictionlogit_tuned, x.evaluate,
"Logistic Regression (Tuned Model)")
LogitOutput_tuned$PercCorrect <- mean(x.evaluate$correctlogit_tuned)*100
Logitconfmatrix_tuned <- table(x.evaluate$predictionlogitclass_tuned,
x.evaluate$ch_in_string)
LogitOutput_tuned$TimeElapsed <- tuned_time[3]

### EVALUATION

print("")
print(" # Model 1: Logistic Regression (Base)")
print("Confusion matrix:")
print(Logitconfmatrix_base)

print("")
print(" # Model 1: Logistic Regression (Tuned)")
print(paste("Best alpha:", x.modelLogit_tuned$bestTune$alpha))
print(paste("Best lambda:", x.modelLogit_tuned$bestTune$lambda))
print("Confusion matrix:")
print(Logitconfmatrix_tuned)

lg_base <- get_metrics(
  pred = x.evaluate$predictionlogit_base,
  pred_class = x.evaluate$predictionlogitclass_base,
  reference = x.evaluate$ch_in_string,
  tdl = LogitOutput_base$TDL,
  gini = LogitOutput_base$GINI,
  time_elapsed = LogitOutput_base$TimeElapsed
)

lg_tuned <- get_metrics(
```

```
  pred = x.evaluate$predictionlogit_tuned,
  pred_class = x.evaluate$predictionlogitclass_tuned,
  reference = x.evaluate$ch_in_string,
  tdl = LogitOutput_tuned$TDL,
  gini = LogitOutput_tuned$GINI,
  time_elapsed = LogitOutput_tuned$TimeElapsed
)

metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_df <- data.frame(
  Metrics = metrics,
  Base = as.numeric(lg_base),
  Tuned = as.numeric(lg_tuned)
)

stargazer(
  results_df,
  summary = FALSE, rownames = FALSE,
  title = "Logistic Regression Model Performance",
  type = "text", digits = 3
)

# the best tuned model returns performance metrics that are relatively close to
the base model
# Recall and Accuracy dropped significantly, meanwhile Specificity increased
dramatically
# therefore, considering the trade-offs, the best model for LG is base model

Logit <- x.modelLogit_base
LogitOutput <- LogitOutput_base
Logitconfmatrix <- Logitconfmatrix_base
logit <- lg_base
x.evaluate$predictionlogit <- x.evaluate$predictionlogit_base
predictionlogit <- x.evaluate$predictionlogit_base



############ 2) NAIVE BAYES ------------------------------------------------

### BASE MODEL

ptm_nb_base <- proc.time()

set.seed(123)
x.modelNB_base <- train(BaseFormula_dum, data = x.trainnorm,
                        method = "naive_bayes", trControl = ctrl)

nb_base_time <- proc.time() - ptm_nb_base

x.evaluate$predictionNB_base <- predict(x.modelNB_base, newdata = x.evaluatenorm,
type = "prob")
```

```
x.evaluate$predictionNBclass_base <-
ifelse(x.evaluate$predictionNB_base[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$correctNB_base <- x.evaluate$predictionNBclass_base ==
x.evaluate$ch_in_string

NBOutput_base <- makeLiftPlot(x.evaluate$predictionNB_base[,"ch_in"], x.evaluate,
"Naive Bayes (Base Model)")
NBOutput_base$PercCorrect <- mean(x.evaluate$correctNB_base)*100
NBconfmatrix_base <- table(x.evaluate$predictionNBclass_base,
x.evaluate$ch_in_string)
NBOutput_base$TimeElapsed <- nb_base_time[3]

### HYPERPARAMETER TUNING

grid_nb <- expand.grid(
  laplace   = 10,
  usekernel = TRUE,
  adjust    = 1
)

ptm_nb_tune <- proc.time()

set.seed(123)
x.modelNB_tuned <- train(
  BaseFormula_dum, data = x.trainnorm, method = "naive_bayes",
  trControl = ctrl,
  tuneGrid = grid_nb
)
nb_tuned_time <- proc.time() - ptm_nb_tune

x.evaluate$predictionNB_tuned <- predict(x.modelNB_tuned, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionNBclass_tuned <-
ifelse(x.evaluate$predictionNB_tuned[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$correctNB_tuned <- x.evaluate$predictionNBclass_tuned ==
x.evaluate$ch_in_string

NBOutput_tuned <- makeLiftPlot(x.evaluate$predictionNB_tuned[,"ch_in"],
x.evaluate, "Naive Bayes (Tuned Model)")
NBOutput_tuned$PercCorrect <- mean(x.evaluate$correctNB_tuned)*100
NBconfmatrix_tuned <- table(x.evaluate$predictionNBclass_tuned,
x.evaluate$ch_in_string)
NBOutput_tuned$TimeElapsed <- nb_tuned_time[3]

### EVALUATION

print("")
print(" # Model 2: Naive Bayes (Base)")
print("Confusion matrix:")
print(NBconfmatrix_base)

print("")
```

```
print(" # Model 2: Naive Bayes (Tuned)")
print(paste("Best laplace:", x.modelNB_tuned$bestTune$laplace))
print(paste("Best usekernel:", x.modelNB_tuned$bestTune$usekernel))
print(paste("Best adjust:", x.modelNB_tuned$bestTune$adjust))
print("Confusion matrix:")
print(NBconfmatrix_tuned)

nb_base <- get_metrics(
  pred = x.evaluate$predictionNB_base,
  pred_class = x.evaluate$predictionNBclass_base,
  reference = x.evaluate$ch_in_string,
  tdl = NBOutput_base$TDL,
  gini = NBOutput_base$GINI,
  time_elapsed = NBOutput_base$TimeElapsed
)

nb_tuned <- get_metrics(
  pred = x.evaluate$predictionNB_tuned,
  pred_class = x.evaluate$predictionNBclass_tuned,
  reference = x.evaluate$ch_in_string,
  tdl = NBOutput_tuned$TDL,
  gini = NBOutput_tuned$GINI,
  time_elapsed = NBOutput_tuned$TimeElapsed
)

metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_df <- data.frame(
  Metrics = metrics,
  Base = as.numeric(nb_base),
  Tuned = as.numeric(nb_tuned)
)

stargazer(
  results_df,
  summary = FALSE, rownames = FALSE,
  title = "Naive Bayes Model Performance",
  type = "text", digits = 3
)

# the best tuned model returns performance metrics that are exactly same as base
model
# additionally, the base model achieves relatively good performance already
# therefore, the best model for NB is base model

NB <- x.modelNB_base
NBOutput <-NBOutput_base
NBconfmatrix <- NBconfmatrix_base
nb <- nb_base
x.evaluate$predictionNB <- x.evaluate$predictionNB_base
predictionNB <- x.evaluate$predictionNB_base
```

```
############ 3) K-NEAREST NEIGHBORS (KNN) ------------------------------------

### BASE MODEL

ptm_knn_base <- proc.time()

set.seed(123)
x.modelKNN_base <- train(
  BaseFormula_dum, data = x.trainnorm, method = "knn",
  trControl = ctrl
)

knn_base_time <- proc.time() - ptm_knn_base

x.evaluate$predictionKNN_base <- predict(x.modelKNN_base, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionKNNclass_base <-
ifelse(x.evaluate$predictionKNN_base[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$correctKNN_base <- x.evaluate$predictionKNNclass_base ==
x.evaluate$ch_in_string

KNNOutput_base <- makeLiftPlot(x.evaluate$predictionKNN_base[,"ch_in"],
x.evaluate, "KNN (Base Model)")
KNNOutput_base$PercCorrect <- mean(x.evaluate$correctKNN_base)*100
KNNconfmatrix_base <- table(x.evaluate$predictionKNNclass_base,
x.evaluate$ch_in_string)
KNNOutput_base$TimeElapsed <- knn_base_time[3]

### HYPERPARAMETER TUNING

grid_knn <- expand.grid(k = 25)

ptm_knn_tune <- proc.time()

set.seed(123)
x.modelKNN_tuned <- train(
  BaseFormula_dum, data = x.trainnorm, method = "knn",
  trControl = ctrl,
  tuneGrid = grid_knn
)
knn_tuned_time <- proc.time() - ptm_knn_tune

x.evaluate$predictionKNN_tuned <- predict(x.modelKNN_tuned, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionKNNclass_tuned <-
ifelse(x.evaluate$predictionKNN_tuned[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$correctKNN_tuned <- x.evaluate$predictionKNNclass_tuned ==
x.evaluate$ch_in_string
```

```
KNNOutput_tuned <- makeLiftPlot(x.evaluate$predictionKNN_tuned[,"ch_in"],
x.evaluate, "KNN (Tuned Model)")
KNNOutput_tuned$PercCorrect <- mean(x.evaluate$correctKNN_tuned)*100
KNNconfmatrix_tuned <- table(x.evaluate$predictionKNNclass_tuned,
x.evaluate$ch_in_string)
KNNOutput_tuned$TimeElapsed <- knn_tuned_time[3]

### EVALUATION

print("")
print(" # Model 3: KNN (Base)")
print("Confusion matrix:")
print(KNNconfmatrix_base)

print("")
print(" # Model 3: KNN (Tuned)")
print(paste("Best k:", x.modelKNN_tuned$bestTune$k))
print("Confusion matrix:")
print(KNNconfmatrix_tuned)

knn_base <- get_metrics(
  pred = x.evaluate$predictionKNN_base,
  pred_class = x.evaluate$predictionKNNclass_base,
  reference = x.evaluate$ch_in_string,
  tdl = KNNOutput_base$TDL,
  gini = KNNOutput_base$GINI,
  time_elapsed = KNNOutput_base$TimeElapsed
)

knn_tuned <- get_metrics(
  pred = x.evaluate$predictionKNN_tuned,
  pred_class = x.evaluate$predictionKNNclass_tuned,
  reference = x.evaluate$ch_in_string,
  tdl = KNNOutput_tuned$TDL,
  gini = KNNOutput_tuned$GINI,
  time_elapsed = KNNOutput_tuned$TimeElapsed
)

metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_df <- data.frame(
  Metrics = metrics,
  Base = as.numeric(knn_base),
  Tuned = as.numeric(knn_tuned)
)

stargazer(
  results_df,
  summary = FALSE, rownames = FALSE,
  title = "KNN Model Performance",
  type = "text", digits = 3
```

```
)

# the best tuned model performs better than the base model in all important
metrics (TDL, GINI, AUC, Precision)
# only 1.6% dropped in F1 due to 3.2% dropped in Recall but Precision relatively
stays the same (increased only 0.3%)
# Specificity increased double
# therefore, the best model for KNN is tuned model

KNN <- x.modelKNN_tuned
KNNOutput <- KNNOutput_tuned
KNNconfmatrix <- KNNconfmatrix_tuned
knn <- knn_tuned
x.evaluate$predictionKNN <- x.evaluate$predictionKNN_tuned
predictionKNN <- x.evaluate$predictionKNN_tuned



############ 4) SUPPORT VECTOR MACHINE (SVM) ---------------------------------

### BASE MODEL

ptm_svm_base <- proc.time()

set.seed(123)
x.modelSVM_base <- train(
  BaseFormula_dum, data = x.trainnorm, method = "svmRadial",
  cachesize=12000, tolerance=.01,
  trControl = ctrl
)

svm_base_time <- proc.time() - ptm_svm_base

x.evaluate$predictionSVM_base <- predict(x.modelSVM_base, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionSVMclass_base <-
ifelse(x.evaluate$predictionSVM_base[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$correctSVM_base <- x.evaluate$predictionSVMclass_base ==
x.evaluate$ch_in_string

SVMOutput_base <- makeLiftPlot(x.evaluate$predictionSVM_base[,"ch_in"],
x.evaluate, "SVM (Base Model)")
SVMOutput_base$PercCorrect <- mean(x.evaluate$correctSVM_base)*100
SVMconfmatrix_base <- table(x.evaluate$predictionSVMclass_base,
x.evaluate$ch_in_string)
SVMOutput_base$TimeElapsed <- svm_base_time[3]

### HYPERPARAMETER TUNING

svm_grid <- expand.grid(
  C = 2,
  sigma = 0.2
)
```

```
ptm_svm_tune <- proc.time()

set.seed(123)
x.modelSVM_tuned <- train(
  BaseFormula_dum, data = x.trainnorm, method = "svmRadial",
  cachesize=12000, tolerance=.01,
  trControl = ctrl,
  tuneGrid = svm_grid
)
svm_tuned_time <- proc.time() - ptm_svm_tune

x.evaluate$predictionSVM_tuned <- predict(x.modelSVM_tuned, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionSVMclass_tuned <-
ifelse(x.evaluate$predictionSVM_tuned[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$correctSVM_tuned <- x.evaluate$predictionSVMclass_tuned ==
x.evaluate$ch_in_string

SVMOutput_tuned <- makeLiftPlot(x.evaluate$predictionSVM_tuned[,"ch_in"],
x.evaluate, "SVM (Tuned Model)")
SVMOutput_tuned$PercCorrect <- mean(x.evaluate$correctSVM_tuned)*100
SVMconfmatrix_tuned <- table(x.evaluate$predictionSVMclass_tuned,
x.evaluate$ch_in_string)
SVMOutput_tuned$TimeElapsed <- svm_tuned_time[3]

### EVALUATION

print("")
print(" # Model 4: SVM (Base)")
print("Confusion matrix:")
print(SVMconfmatrix_base)

print("")
print(" # Model 4: SVM (Tuned)")
print(paste("Best C:", x.modelSVM_tuned$bestTune$C))
print(paste("Best sigma:", x.modelSVM_tuned$bestTune$sigma))
print("Confusion matrix:")
print(SVMconfmatrix_tuned)

svm_base <- get_metrics(
  pred = x.evaluate$predictionSVM_base,
  pred_class = x.evaluate$predictionSVMclass_base,
  reference = x.evaluate$ch_in_string,
  tdl = SVMOutput_base$TDL,
  gini = SVMOutput_base$GINI,
  time_elapsed = SVMOutput_base$TimeElapsed
)

svm_tuned <- get_metrics(
  pred = x.evaluate$predictionSVM_tuned,
  pred_class = x.evaluate$predictionSVMclass_tuned,
```

```
  reference = x.evaluate$ch_in_string,
  tdl = SVMOutput_tuned$TDL,
  gini = SVMOutput_tuned$GINI,
  time_elapsed = SVMOutput_tuned$TimeElapsed
)

metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_df <- data.frame(
  Metrics = metrics,
  Base = as.numeric(svm_base),
  Tuned = as.numeric(svm_tuned)
)

stargazer(
  results_df,
  summary = FALSE, rownames = FALSE,
  title = "SVM Model Performance",
  type = "text", digits = 3
)

# the more tuning, the worse the tuned model gets
# all metrics of tuned model become lower except F1, Recall, Accuracy but those
are not the most important metrics
# therefore, the best model for SVM is the base model

SVM <- x.modelSVM_base
SVMOutput <- SVMOutput_base
SVMconfmatrix <- SVMconfmatrix_base
svm <- svm_base
x.evaluate$predictionSVM <- x.evaluate$predictionSVM_base
predictionSVM <- x.evaluate$predictionSVM_base



########## 5) DECISION TREE ------------------------------------------------

### BASE MODEL

ptm_tree_base <- proc.time()

set.seed(123)
x.modelTree_base <- train(
  BaseFormula_dum, data = x.trainnorm, method = "ctree",
  trControl = ctrl
)

tree_base_time <- proc.time() - ptm_tree_base

x.evaluate$predictionTree_base <- predict(x.modelTree_base, newdata =
x.evaluatenorm, type = "prob")
```

```
x.evaluate$predictionTreeClass_base <-
ifelse(x.evaluate$predictionTree_base[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$predictionTreeClass_base <-
factor(x.evaluate$predictionTreeClass_base, levels = c("Noch_in", "ch_in"))
x.evaluate$correctTree_base <- x.evaluate$predictionTreeClass_base ==
x.evaluate$ch_in_string

TreeOutput_base <- makeLiftPlot(x.evaluate$predictionTree_base[,"ch_in"],
x.evaluate, "Decision Tree (Base)")
TreeOutput_base$PercCorrect <- mean(x.evaluate$correctTree_base) * 100
Treeconfmatrix_base <- table(x.evaluate$predictionTreeClass_base,
x.evaluate$ch_in_string)
TreeOutput_base$TimeElapsed <- tree_base_time[3]

### HYPERPARAMETER TUNING

grid_dt <- expand.grid(cp = seq(0.0001, 0.5, by = 0.01))

ptm_tree_tune <- proc.time()

set.seed(123)
x.modelTree_tuned <- train(
  BaseFormula_dum, data = x.trainnorm, method = "rpart",
  trControl = ctrl,
  tuneGrid = grid_dt,
  control = rpart.control(minsplit = 1, maxdepth = 5)
)

tree_tuned_time <- proc.time() - ptm_tree_tune

x.evaluate$predictionTree_tuned <- predict(x.modelTree_tuned, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionTreeClass_tuned <-
ifelse(x.evaluate$predictionTree_tuned[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$predictionTreeClass_tuned <-
factor(x.evaluate$predictionTreeClass_tuned, levels = c("Noch_in", "ch_in"))
x.evaluate$correctTree_tuned <- x.evaluate$predictionTreeClass_tuned ==
x.evaluate$ch_in_string

TreeOutput_tuned <- makeLiftPlot(x.evaluate$predictionTree_tuned[,"ch_in"],
x.evaluate, "Decision Tree (Tuned)")
TreeOutput_tuned$PercCorrect <- mean(x.evaluate$correctTree_tuned) * 100
Treeconfmatrix_tuned <- table(x.evaluate$predictionTreeClass_tuned,
x.evaluate$ch_in_string)
TreeOutput_tuned$TimeElapsed <- tree_tuned_time[3]

### EVALUATION

print("")
print(" # Model 5: Decision Tree (Base)")
print("Confusion matrix:")
print(Treeconfmatrix_base)
```

```
print("")
print(" # Model 5: Decision Tree (Tuned)")
print(paste("Best cp:", x.modelTree_tuned$bestTune$cp))
print("Confusion matrix:")
print(Treeconfmatrix_tuned)

tree_base <- get_metrics(
  pred = x.evaluate$predictionTree_base,
  pred_class = x.evaluate$predictionTreeClass_base,
  reference = x.evaluate$ch_in_string,
  tdl = TreeOutput_base$TDL,
  gini = TreeOutput_base$GINI,
  time_elapsed = TreeOutput_base$TimeElapsed
)

tree_tuned <- get_metrics(
  pred = x.evaluate$predictionTree_tuned,
  pred_class = x.evaluate$predictionTreeClass_tuned,
  reference = x.evaluate$ch_in_string,
  tdl = TreeOutput_tuned$TDL,
  gini = TreeOutput_tuned$GINI,
  time_elapsed = TreeOutput_tuned$TimeElapsed
)

metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_df <- data.frame(
  Metrics = metrics,
  Base = as.numeric(tree_base),
  Tuned = as.numeric(tree_tuned)
)

stargazer(
  results_df,
  summary = FALSE, rownames = FALSE,
  title = "Decision Tree Model Performance",
  type = "text", digits = 3
)

# the best tuned model performs significantly better than the base model in all
important metrics
# TDL increase 0.5, GINI increased 0.3, AUC increased 0.2, F1 decreased slighly
due to 3% dropped in Recall
# Accuracy decreased 4% meanwhile Specificity increased 6%
# therefore, the best model for DT is the tuned model

Tree <- x.modelTree_tuned
TreeOutput <- TreeOutput_tuned
Treeconfmatrix <- TreeOutput_tuned
tree <- tree_tuned
```

```
x.evaluate$predictionTree <- x.evaluate$predictionTree_tuned
predictionTree <- x.evaluate$predictionTree_tuned



############ 6) RANDOM FOREST ------------------------------------------------

# BASE MODEL

ptm_rf_base <- proc.time()

set.seed(123)
x.modelRF_base <- train(
  BaseFormula_dum, data = x.trainnorm, method = "parRF",
  trControl = ctrl
)

rf_base_time <- proc.time() - ptm_rf_base

x.evaluate$predictionRF_base <- predict(x.modelRF_base, newdata = x.evaluatenorm,
type = "prob")
x.evaluate$predictionRFClass_base <-
ifelse(x.evaluate$predictionRF_base[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$predictionRFClass_base <- factor(x.evaluate$predictionRFClass_base,
levels = c("Noch_in", "ch_in"))
x.evaluate$correctRF_base <- x.evaluate$predictionRFClass_base ==
x.evaluate$ch_in_string

RFOutput_base <- makeLiftPlot(x.evaluate$predictionRF_base[,'ch_in'], x.evaluate,
"Random Forest (Base)")
RFOutput_base$PercCorrect <- mean(x.evaluate$correctRF_base) * 100
RFconfmatrix_base <- table(x.evaluate$predictionRFClass_base,
x.evaluate$ch_in_string)
RFOutput_base$TimeElapsed <- rf_base_time[3]

# HYPERPARAMETER TUNING

grid_rf <- expand.grid(
  mtry = floor(seq(2, sqrt(ncol(x.trainnorm) - 2), length.out = 3)),
  splitrule = c("gini", "extratrees"),
  min.node.size = c(1, 5, 10)
)

ptm_rf_tune <- proc.time()

set.seed(123)
x.modelRF_tuned <- train(
  BaseFormula_dum, data = x.trainnorm, method = "ranger",
  trControl = ctrl,
  tuneGrid = grid_rf,
  num.trees = 100,
  max.depth = 10,
  importance = "impurity"
```

```
)

rf_tuned_time <- proc.time() - ptm_rf_tune

x.evaluate$predictionRF_tuned <- predict(x.modelRF_tuned, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionRFClass_tuned <-
ifelse(x.evaluate$predictionRF_tuned[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$predictionRFClass_tuned <- factor(x.evaluate$predictionRFClass_tuned,
levels = c("Noch_in", "ch_in"))
x.evaluate$correctRF_tuned <- x.evaluate$predictionRFClass_tuned ==
x.evaluate$ch_in_string

RFOutput_tuned <- makeLiftPlot(x.evaluate$predictionRF_tuned[,'ch_in'],
x.evaluate, "Random Forest (Tuned)")
RFOutput_tuned$PercCorrect <- mean(x.evaluate$correctRF_tuned) * 100
RFconfmatrix_tuned <- table(x.evaluate$predictionRFClass_tuned,
x.evaluate$ch_in_string)
RFOutput_tuned$TimeElapsed <- rf_tuned_time[3]

# EVALUATION

print("")
print(" # Model 6: Random Forest (Base)")
print("Confusion matrix:")
print(RFconfmatrix_base)

print("")
print(" # Model 6: Random Forest (Tuned)")
print(paste("Best mtry:", x.modelRF_tuned$bestTune$mtry))
print(paste("Best splitrule:", x.modelRF_tuned$bestTune$splitrule))
print(paste("Best min.node.size:", x.modelRF_tuned$bestTune$min.node.size))
print("Confusion matrix:")
print(RFconfmatrix_tuned)

rf_base <- get_metrics(
  pred = x.evaluate$predictionRF_base,
  pred_class = x.evaluate$predictionRFClass_base,
  reference = x.evaluate$ch_in_string,
  tdl = RFOutput_base$TDL,
  gini = RFOutput_base$GINI,
  time_elapsed = RFOutput_base$TimeElapsed
)

rf_tuned <- get_metrics(
  pred = x.evaluate$predictionRF_tuned,
  pred_class = x.evaluate$predictionRFClass_tuned,
  reference = x.evaluate$ch_in_string,
  tdl = RFOutput_tuned$TDL,
  gini = RFOutput_tuned$GINI,
  time_elapsed = RFOutput_tuned$TimeElapsed
)
```

```
metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_df <- data.frame(
  Metrics = metrics,
  Base = as.numeric(rf_base),
  Tuned = as.numeric(rf_tuned)
)

stargazer(
  results_df,
  summary = FALSE, rownames = FALSE,
  title = "Random Forest Model Performance",
  type = "text", digits = 3
)

# the best tuned model performs significantly better than the base model in terms
of TDL
# GINI, AUC, Precision, Specificity are also higher after tuning
# the tuned model takes less time to run than the base model
# therefore, the best model for RF is tuned model

RF <- x.modelRF_tuned
RFOutput <- RFOutput_tuned
RFconfmatrix <- RFconfmatrix_tuned
rf <- rf_tuned
x.evaluate$predictionRF <- x.evaluate$predictionRF_tuned
predictionRF <- x.evaluate$predictionRF_tuned



############ 7) BAGGING ----------------------------------------------------

# BASE MODEL

ptm_bag_base <- proc.time()

set.seed(123)
x.modelBagging_base <- train(
  BaseFormula_dum,
  data = x.trainnorm,
  method = "treebag",
  trControl = ctrl
)

bag_base_time <- proc.time() - ptm_bag_base

x.evaluate$predictionBagging_base <- predict(x.modelBagging_base, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionBaggingClass_base <-
ifelse(x.evaluate$predictionBagging_base[,"ch_in"] > probthres, "ch_in",
"Noch_in")
```

```
x.evaluate$predictionBaggingClass_base <-
factor(x.evaluate$predictionBaggingClass_base, levels = c("Noch_in", "ch_in"))
x.evaluate$correctBagging_base <- x.evaluate$predictionBaggingClass_base ==
x.evaluate$ch_in_string

BaggingOutput_base <- makeLiftPlot(x.evaluate$predictionBagging_base[,'ch_in'],
x.evaluate, "Bagging (Base)")
BaggingOutput_base$PercCorrect <- mean(x.evaluate$correctBagging_base) * 100
Baggingconfmatrix_base <- table(x.evaluate$predictionBaggingClass_base,
x.evaluate$ch_in_string)
BaggingOutput_base$TimeElapsed <- bag_base_time[3]

# HYPERPARAMETER TUNING

#grid_bag <- expand.grid(nbagg = c(10, 25, 50),
                        #maxdepth = c(3, 5, 7),
                        #minsplit = c(5, 10, 20))

ptm_bag_tune <- proc.time()

set.seed(123)
x.modelBagging_tuned <- train(
  BaseFormula_dum,
  data = x.trainnorm,
  method = "treebag",
  trControl = ctrl,
  nbagg = 100,
  #tuneGrid = grid_bag
)

bag_tuned_time <- proc.time() - ptm_bag_tune

x.evaluate$predictionBagging_tuned <- predict(x.modelBagging_tuned, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionBaggingClass_tuned <-
ifelse(x.evaluate$predictionBagging_tuned[,"ch_in"] > probthres, "ch_in",
"Noch_in")
x.evaluate$predictionBaggingClass_tuned <-
factor(x.evaluate$predictionBaggingClass_tuned, levels = c("Noch_in", "ch_in"))
x.evaluate$correctBagging_tuned <- x.evaluate$predictionBaggingClass_tuned ==
x.evaluate$ch_in_string

BaggingOutput_tuned <- makeLiftPlot(x.evaluate$predictionBagging_tuned[,'ch_in'],
x.evaluate, "Bagging (Tuned)")
BaggingOutput_tuned$PercCorrect <- mean(x.evaluate$correctBagging_tuned) * 100
Baggingconfmatrix_tuned <- table(x.evaluate$predictionBaggingClass_tuned,
x.evaluate$ch_in_string)
BaggingOutput_tuned$TimeElapsed <- bag_tuned_time[3]

# EVALUATION

print("")
```

```
print(" # Model 7: Bagging (Base)")
print("Confusion matrix:")
print(Baggingconfmatrix_base)

print("")
print(" # Model 7: Bagging (Tuned)")
print(paste("Best nbagg:", x.modelBagging_tuned$bestTune$nbagg))
print("Confusion matrix:")
print(Baggingconfmatrix_tuned)

bag_base <- get_metrics(
  pred = x.evaluate$predictionBagging_base,
  pred_class = x.evaluate$predictionBaggingClass_base,
  reference = x.evaluate$ch_in_string,
  tdl = BaggingOutput_base$TDL,
  gini = BaggingOutput_base$GINI,
  time_elapsed = BaggingOutput_base$TimeElapsed
)

bag_tuned <- get_metrics(
  pred = x.evaluate$predictionBagging_tuned,
  pred_class = x.evaluate$predictionBaggingClass_tuned,
  reference = x.evaluate$ch_in_string,
  tdl = BaggingOutput_tuned$TDL,
  gini = BaggingOutput_tuned$GINI,
  time_elapsed = BaggingOutput_tuned$TimeElapsed
)

metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_bagging <- data.frame(
  Metrics = metrics,
  Base = as.numeric(bag_base),
  Tuned = as.numeric(bag_tuned)
)

stargazer(
  results_bagging,
  summary = FALSE, rownames = FALSE,
  title = "Bagging Model Performance",
  type = "text", digits = 3
)

# the tuned model performs better in all metrics except TDL
# TDL dropped 0.3, while GINI increased 0.2, AUC increased 0.1
# all other metrics increased but the difference is relatively small (~0.02%)
# tunning take significantly longer time
# therefore, the best model for Bagging is the base model

Bagging <- x.modelBagging_base
BaggingOutput <- BaggingOutput_base
```

```
Baggingconfmatrix <- Baggingconfmatrix_base
bagging <- bag_base
x.evaluate$predictionBagging <- x.evaluate$predictionBagging_base
predictionBagging <- x.evaluate$predictionBagging_base



############ 8) BOOSTING ----------------------------------------------------

# BASE MODEL
ptm_xgb_base <- proc.time()

set.seed(123)
x.modelXGB_base <- train(
  BaseFormula_dum, data = x.trainnorm, method = 'blackboost',
  trControl = ctrl
)

xgb_base_time <- proc.time() - ptm_xgb_base

x.evaluate$predictionXGB_base <- predict(x.modelXGB_base, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionXGBClass_base <-
ifelse(x.evaluate$predictionXGB_base[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$predictionXGBClass_base <- factor(x.evaluate$predictionXGBClass_base,
levels = c("Noch_in", "ch_in"))
x.evaluate$correctXGB_base <- x.evaluate$predictionXGBClass_base ==
x.evaluate$ch_in_string

XGBOutput_base <- makeLiftPlot(x.evaluate$predictionXGB_base[,'ch_in'],
x.evaluate, "Boosting (Base)")
XGBOutput_base$PercCorrect <- mean(x.evaluate$correctXGB_base) * 100
XGBconfmatrix_base <- table(x.evaluate$predictionXGBClass_base,
x.evaluate$ch_in_string)
XGBOutput_base$TimeElapsed <- xgb_base_time[3]

# HYPERPARAMETER TUNING

grid_xgb <- expand.grid(
  nrounds = 150,
  max_depth = 2,
  eta = 0.1,
  gamma = 0,
  colsample_bytree = 1,
  min_child_weight = 1,
  subsample = 1
)

ptm_xgb_tune <- proc.time()

set.seed(123)
x.modelXGB_tuned <- train(
```

```
  BaseFormula_dum, data = x.trainnorm, method = "xgbTree",
  trControl = ctrl,
  tuneGrid = grid_xgb,
  verbose = FALSE
)

xgb_tuned_time <- proc.time() - ptm_xgb_tune

x.evaluate$predictionXGB_tuned <- predict(x.modelXGB_tuned, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionXGBClass_tuned <-
ifelse(x.evaluate$predictionXGB_tuned[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$predictionXGBClass_tuned <-
factor(x.evaluate$predictionXGBClass_tuned, levels = c("Noch_in", "ch_in"))
x.evaluate$correctXGB_tuned <- x.evaluate$predictionXGBClass_tuned ==
x.evaluate$ch_in_string

XGBOutput_tuned <- makeLiftPlot(x.evaluate$predictionXGB_tuned[,'ch_in'],
x.evaluate, "Boosting (Tuned)")
XGBOutput_tuned$PercCorrect <- mean(x.evaluate$correctXGB_tuned) * 100
XGBconfmatrix_tuned <- table(x.evaluate$predictionXGBClass_tuned,
x.evaluate$ch_in_string)
XGBOutput_tuned$TimeElapsed <- xgb_tuned_time[3]

# EVALUATION

print("")
print(" # Model 8: Boosting (Base)")
print("Confusion matrix:")
print(XGBconfmatrix_base)

print("")
print(" # Model 8: Boosting (Tuned)")
print(paste("Best nrounds:", x.modelXGB_tuned$bestTune$nrounds))
print(paste("Best max_depth:", x.modelXGB_tuned$bestTune$max_depth))
print(paste("Best eta:", x.modelXGB_tuned$bestTune$eta))
print(paste("Best gamma:", x.modelXGB_tuned$bestTune$gamma))
print(paste("Best colsample_bytree:",
x.modelXGB_tuned$bestTune$colsample_bytree))
print(paste("Best min_child_weight:",
x.modelXGB_tuned$bestTune$min_child_weight))
print(paste("Best subsample:", x.modelXGB_tuned$bestTune$subsample))
print("Confusion matrix:")
print(XGBconfmatrix_tuned)

xgb_base <- get_metrics(
  pred = x.evaluate$predictionXGB_base,
  pred_class = x.evaluate$predictionXGBClass_base,
  reference = x.evaluate$ch_in_string,
  tdl = XGBOutput_base$TDL,
  gini = XGBOutput_base$GINI,
  time_elapsed = XGBOutput_base$TimeElapsed
```

```
)

xgb_tuned <- get_metrics(
  pred = x.evaluate$predictionXGB_tuned,
  pred_class = x.evaluate$predictionXGBClass_tuned,
  reference = x.evaluate$ch_in_string,
  tdl = XGBOutput_tuned$TDL,
  gini = XGBOutput_tuned$GINI,
  time_elapsed = XGBOutput_tuned$TimeElapsed
)

metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_df <- data.frame(
  Metrics = metrics,
  Base = as.numeric(xgb_base),
  Tuned = as.numeric(xgb_tuned)
)

stargazer(
  results_df,
  summary = FALSE, rownames = FALSE,
  title = "Boosting Model Performance",
  type = "text", digits = 3
)

# the bets tuned model performs slightly better than the base model
# TDL dropped 0.2, GINI and AUC increased ~0.02
# F1 improved 5% but due to  8% increase in Recall while Precision stays
relatively same
# Accuracy increased 7%
# Specificity dropped significantly but it is not an important metric
# therefore, the best model for Boosting is the base model

Boosting <- x.modelXGB_base
BoostingOutput <- XGBOutput_base
Boostingconfmatrix <- XGBconfmatrix_base
boosting <- xgb_base
x.evaluate$predictionBoosting <- x.evaluate$predictionXGB_base
predictionBoosting <- x.evaluate$predictionXGB_base



########### 9) NEURAL NETWORK ------------------------------------------------

# BASE MODEL
ptm_nnet_base <- proc.time()

set.seed(123)

mlp_grid = expand.grid(layer1 = 5,
```

```
                        layer2 = 5,
                        layer3 = 0)
x.modelNNet_base <- train(BaseFormula_dum, data=x.trainnorm,
method='mlpML',tuneGrid=mlp_grid)

nnet_base_time <- proc.time() - ptm_nnet_base

# plot NNet
if(0){
  NeuralNetTools::plotnet(x.modelNNet_base$finalModel)
}

x.evaluate$predictionNNet_base <- predict(x.modelNNet_base, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionNNetClass_base <-
ifelse(x.evaluate$predictionNNet_base[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$predictionNNetClass_base <-
factor(x.evaluate$predictionNNetClass_base, levels = c("Noch_in", "ch_in"))
x.evaluate$correctNNet_base <- x.evaluate$predictionNNetClass_base ==
x.evaluate$ch_in_string

NNetOutput_base <- makeLiftPlot(x.evaluate$predictionNNet_base[,'ch_in'],
x.evaluate, "Neural Network (Base)")
NNetOutput_base$PercCorrect <- mean(x.evaluate$correctNNet_base) * 100
NNetconfmatrix_base <- table(x.evaluate$predictionNNetClass_base,
x.evaluate$ch_in_string)
NNetOutput_base$TimeElapsed <- nnet_base_time[3]

# HYPERPARAMETER TUNING

# grid_nnet <- expand.grid(
#   size = c(3, 5, 7),
#   decay = c(0, 0.01, 0.1)
# )

ptm_nnet_tune <- proc.time()

grid_nnet <- expand.grid(
  layer1 = 30,
  layer2 = 2,
  layer3 = 0
)

set.seed(123)
x.modelNNet_tuned <- train(
  BaseFormula_dum, data = x.trainnorm, method = "mlpML",
  trControl = ctrl,
  tuneGrid = grid_nnet,
  maxit = 200,
  trace = FALSE
)
```

```
# set.seed(123)
# x.modelNNet_tuned <- train(
#   BaseFormula_dum, data = x.trainnorm, method = "nnet",
#   trControl = ctrl,
#   tuneGrid = grid_nnet,
#   maxit = 200,
#   trace = FALSE
# )

nnet_tuned_time <- proc.time() - ptm_nnet_tune

# plot NNet
if(0){
  NeuralNetTools::plotnet(x.modelNNet_tuned$finalModel)
}

x.evaluate$predictionNNet_tuned <- predict(x.modelNNet_tuned, newdata =
x.evaluatenorm, type = "prob")
x.evaluate$predictionNNetClass_tuned <-
ifelse(x.evaluate$predictionNNet_tuned[,"ch_in"] > probthres, "ch_in", "Noch_in")
x.evaluate$predictionNNetClass_tuned <-
factor(x.evaluate$predictionNNetClass_tuned, levels = c("Noch_in", "ch_in"))
x.evaluate$correctNNet_tuned <- x.evaluate$predictionNNetClass_tuned ==
x.evaluate$ch_in_string

NNetOutput_tuned <- makeLiftPlot(x.evaluate$predictionNNet_tuned[,'ch_in'],
x.evaluate, "Neural Network (Tuned)")
NNetOutput_tuned$PercCorrect <- mean(x.evaluate$correctNNet_tuned) * 100
NNetconfmatrix_tuned <- table(x.evaluate$predictionNNetClass_tuned,
x.evaluate$ch_in_string)
NNetOutput_tuned$TimeElapsed <- nnet_tuned_time[3]

# EVALUATION

print("")
print(" # Model 9: Neural Network (Base)")
print("Confusion matrix:")
print(NNetconfmatrix_base)

print("")
print(" # Model 9: Neural Network (Tuned)")
print(paste("Best size:", x.modelNNet_tuned$bestTune$size))
print(paste("Best decay:", x.modelNNet_tuned$bestTune$decay))
print("Confusion matrix:")
print(NNetconfmatrix_tuned)

nnet_base <- get_metrics(
  pred = x.evaluate$predictionNNet_base,
  pred_class = x.evaluate$predictionNNetClass_base,
  reference = x.evaluate$ch_in_string,
  tdl = NNetOutput_base$TDL,
  gini = NNetOutput_base$GINI,
```

```
  time_elapsed = NNetOutput_base$TimeElapsed
)

nnet_tuned <- get_metrics(
  pred = x.evaluate$predictionNNet_tuned,
  pred_class = x.evaluate$predictionNNetClass_tuned,
  reference = x.evaluate$ch_in_string,
  tdl = NNetOutput_tuned$TDL,
  gini = NNetOutput_tuned$GINI,
  time_elapsed = NNetOutput_tuned$TimeElapsed
)

metrics <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")

results_df <- data.frame(
  Metrics = metrics,
  Base = as.numeric(nnet_base),
  Tuned = as.numeric(nnet_tuned)
)

stargazer(
  results_df,
  summary = FALSE, rownames = FALSE,
  title = "Neural Network Model Performance",
  type = "text", digits = 3
)

# the best tuned model performs better in all important metrics
# TDL increased significantly 0.6, GINI and AUC increased 0.01, F1 increased 5%
# Precision stays relatively same while Recall increased 10%
# therefore, the best model for NNet is the tuned model

NNet <- x.modelNNet_tuned
NNetOutput <- NNetOutput_tuned
NNetconfmatrix <- NNetconfmatrix_tuned
nnet <- nnet_tuned
x.evaluate$predictionNNet <- x.evaluate$predictionNNet_tuned
predictionNNet <- x.evaluate$predictionNNet_tuned



# performance evaluation-----------------------------------------------------------



x.evaluate$predictionLogit     <- predict(Logit, newdata=x.evaluatedum,
type="prob")[,"ch_in"]
x.evaluate$predictionNB        <- predict(NB, newdata=x.evaluatenorm,
type="prob")[,"ch_in"]
x.evaluate$predictionKNN       <- predict(KNN, newdata=x.evaluatenorm,
type="prob")[,"ch_in"]
```

```
x.evaluate$predictionSVM       <- predict(SVM, newdata=x.evaluatenorm,
type="prob")[,"ch_in"]
x.evaluate$predictionTree      <- predict(Tree, newdata=x.evaluatenorm,
type="prob")[,"ch_in"]
x.evaluate$predictionRF        <- predict(RF, newdata=x.evaluatenorm,
type="prob")[,"ch_in"]
x.evaluate$predictionBagging   <- predict(Bagging, newdata=x.evaluatenorm,
type="prob")[,"ch_in"]
x.evaluate$predictionBoosting  <- predict(Boosting, newdata=x.evaluatenorm,
type="prob")[,"ch_in"]
x.evaluate$predictionNNet      <- predict(NNet, newdata=x.evaluatenorm,
type="prob")[,"ch_in"]

metric_names <- c("TDL", "GINI", "AUC", "F1", "Precision", "Recall", "Accuracy",
"Specificity", "Time")
model_names <- c("LG_FT", "NB_FT", "KNN_FT", "SVM_FT", "DT_FT",
                 "RF_FT", "Bagging_FT", "Boosting_FT", "NNet_FT")


metrics_table <- matrix(NA, nrow=length(metric_names), ncol=length(model_names))
rownames(metrics_table) <- metric_names
colnames(metrics_table) <- model_names

metrics_table[, "LG_FT"]        <- logit
metrics_table[, "NB_FT"]        <- nb
metrics_table[, "KNN_FT"]       <- knn
metrics_table[, "SVM_FT"]       <- svm
metrics_table[, "DT_FT"]        <- tree
metrics_table[, "RF_FT"]        <- rf
metrics_table[, "Bagging_FT"]   <- bagging
metrics_table[, "Boosting_FT"]  <- boosting
metrics_table[, "NNet_FT"]      <- nnet

metrics_table_round <- round(metrics_table, 3)

stargazer(
  metrics_table_round,
  summary = FALSE, rownames = TRUE,
  title = "Model Evaluation",
  type = "text", digits = 3
)

# heat map for performance metrics (with thresholds settings specifically for
each metrics)

metrics_df <- melt(metrics_table_round)
colnames(metrics_df) <- c("Metric", "Model", "Value")

for (i in seq_len(nrow(metrics_df))) {
  metric <- metrics_df$Metric[i]
  val <- metrics_df$Value[i]
  # Adjust scaling for each metric
  if (metric == "TDL") {
```

```
    metrics_df$NormValue[i] <- (val - 1) / (5 - 1)
  } else if (metric %in% c("GINI", "AUC")) {
    metrics_df$NormValue[i] <- (val - 0) / (1 - 0)
  } else if (metric %in% c("F1", "Precision", "Recall", "Accuracy",
"Specificity")) {
    metrics_df$NormValue[i] <- (val - 0) / (100 - 0)
  } else if (metric == "Time") {
    metrics_df$NormValue[i] <- 1 - (val - 0) / (100 - 0)
  } else {
    metrics_df$NormValue[i] <- NA
  }
}

ggplot(metrics_df, aes(x = Model, y = Metric, fill = NormValue)) +
  geom_tile(color = "white", size = 0.5) +
  geom_text(aes(label = Value), color = "black", size = 3) +
  scale_x_discrete(position = "top") +
  scale_y_discrete(limits = rev(unique(metrics_df$Metric))) +
  scale_fill_gradient2(
    low = "red", mid = "lightyellow", high = "lightblue",
    midpoint = 0.5, limits = c(0, 1),
  ) +
  theme_minimal() +
  labs(title = "Fine-Tuned Model Performance") +
  theme(
    axis.text.x = element_text(size = 9),
    axis.text.y = element_text(size = 9),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    plot.title = element_text(size = 14, hjust = 0.5),
    panel.grid = element_blank(),
    legend.position = "none"
  )

# ROC curves

model_list <- list(
  "Logistic Regression" = x.evaluate$predictionlogit,
  "Naive Bayes"         = x.evaluate$predictionNB,
  "KNN"                 = x.evaluate$predictionKNN,
  "SVM"                 = x.evaluate$predictionSVM,
  "Decision Tree"       = x.evaluate$predictionTree,
  "Random Forest"       = x.evaluate$predictionRF,
  "Bagging"             = x.evaluate$predictionBagging,
  "Boosting"            = x.evaluate$predictionBoosting,
  "Neural Network"      = x.evaluate$predictionNNet
)

actual <- x.evaluate$ch_in_string

model_colors <- c(
  "Logistic Regression" = "blue",
```

```
  "Naive Bayes"         = "red",
  "K-Nearest Neighbor"  = "green3",
  "Support Vector Machine"  = "purple",
  "Decision Tree"       = "orange",
  "Random Forest"       = "brown",
  "Bagging"             = "goldenrod",
  "Boosting"            = "darkcyan",
  "Neural Network"      = "black"
)

plot(NULL, xlim=c(1,0), ylim=c(0,1), xlab="False Positive Rate", ylab="True
Positive Rate", main="ROC Curves")
abline(a=0, b=1, lty=2, col="gray")

auc_vals <- numeric(length(model_list))
names(auc_vals) <- names(model_list)

for (model in names(model_list)) {
  probs <- model_list[[model]]
  # Check type:
  if(!is.numeric(probs)) {
    stop(paste0("Predictions for ", model, " are not numeric!"))
  }
  roc_obj <- roc(actual, probs, levels = c("Noch_in", "ch_in"), direction = "<")
  lines(roc_obj, col=model_colors[model], lwd=1)
  auc_vals[model] <- as.numeric(auc(roc_obj))
}

legend_labels <- sprintf("%s (AUC = %.3f)", names(auc_vals), auc_vals)
legend("bottomright", legend=legend_labels, col=model_colors, lwd=1, cex=0.9)

# lift curves

nrow(x.evaluate)
sapply(x.evaluate[, c("predictionlogit", "predictionNB", "predictionKNN",
"predictionSVM",
                      "predictionTree", "predictionRF", "predictionBagging",
"predictionBoosting",
                      "predictionNNet", "ch_in_string")], length)

OverallGINI <- c(LogitOutput$GINI, NBOutput$GINI, KNNOutput$GINI, SVMOutput$GINI,
TreeOutput$GINI, RFOutput$GINI, BaggingOutput$GINI, BoostingOutput$GINI,
NNetOutput$GINI)
OverallTDL  <- c(LogitOutput$TDL, NBOutput$TDL, KNNOutput$TDL, SVMOutput$TDL,
TreeOutput$TDL, RFOutput$TDL, BaggingOutput$TDL, BoostingOutput$TDL,
NNetOutput$TDL)

lift_obj <- lift(ch_in_string ~ predictionlogit + predictionNB + predictionKNN +
predictionSVM +
                   predictionTree + predictionRF + predictionBagging +
predictionBoosting + predictionNNet,
                 data = x.evaluate, class = "ch_in")
```

```
model_names <- c("Logistic Regression", "Naive Bayes", "KNN", "SVM",
                 "Decision Tree", "Random Forest", "Bagging", "Boosting",
                 "Neural Network")

labels_with_metrics <- paste0(model_names, " (GINI: ", round(OverallGINI, 3),
                              ", TDL: ", round(OverallTDL, 3), ")")

ggplot(lift_obj) +
  ggtitle("Lift Curves") +
  scale_color_discrete(name = NULL, labels = labels_with_metrics) +
  theme_minimal() +
  theme(
    plot.title = element_text(
      hjust = 0.5,
      face = "bold",
      color = "black",
      size = 17,
      margin = margin(b = 10)
    ),
    legend.position = c(0.98, 0.02),
    legend.justification = c("right", "bottom"),
    legend.text = element_text(size = 11),
    legend.key.size = unit(0.9, "lines"),
    legend.background = element_rect(fill = scales::alpha("white", 0.8), color =
"gray", size = 0.5)
  )

# TDL & GINI graph

ForGraph <- data.frame(OverallTDL,OverallGINI)

myLeftAxisLabs <- pretty(seq(0, max(ForGraph$OverallTDL), length.out = 10))
myRightAxisLabs <- pretty(seq(0, max(ForGraph$OverallGINI), length.out = 10))

myLeftAxisAt <- myLeftAxisLabs/max(ForGraph$OverallTDL)
myRightAxisAt <- myRightAxisLabs/max(ForGraph$OverallGINI)

ForGraph$OverallTDL1 <- ForGraph$OverallTDL/max(ForGraph$OverallTDL)
ForGraph$OverallGINI1 <- ForGraph$OverallGINI/max(ForGraph$OverallGINI)

op <- par(mar = c(5,4,4,4) + 0.1)

barplot(t(as.matrix(ForGraph[, c("OverallTDL1", "OverallGINI1")])), beside =
TRUE, yaxt = "n",
        names.arg = c("Logistic Regression", "Naive Bayes", "KNN", "SVM",
                      "Decision Tree", "Random Forest", "Bagging", "Boosting",
                      "Neural Network"), ylim=c(0, max(c(myLeftAxisAt,
myRightAxisAt))),
        ylab =  "Top Decile Lift", legend = c("TDL","GINI"), args.legend = list(x
= "topright", inset = c(0, -0.1)),
        main="Performance of the Machine Learning Algorithms")
```

```
axis(2, at = myLeftAxisAt, labels = myLeftAxisLabs)

axis(4, at = myRightAxisAt, labels = myRightAxisLabs)

mtext("GINI Coefficient", side = 4, line = 3, cex = par("cex.lab"))

mtext(c(paste(round(LogitOutput$TimeElapsed,digits=2),"sec"),
        paste(round(KNNOutput$TimeElapsed,digits=2),"sec"),
        paste(round(NBOutput$TimeElapsed,digits=2),"sec"),
        paste(round(SVMOutput$TimeElapsed,digits=2),"sec"),
        paste(round(TreeOutput$TimeElapsed,digits=2),"sec"),
        paste(round(BaggingOutput$TimeElapsed,digits=2),"sec"),
        paste(round(BoostingOutput$TimeElapsed,digits=2),"sec"),
        paste(round(RFOutput$TimeElapsed,digits=2),"sec"),
        paste(round(NNetOutput$TimeElapsed,digits=2),"sec")), side = 1, line = 3,
cex = par("cex.lab"), at = c(2,5,8,11,14,17,20,23,26))
mtext(c(paste(round(LogitOutput$PercCorrect,digits=0),"%"),
        paste(round(KNNOutput$PercCorrect,digits=0),"%"),
        paste(round(NBOutput$PercCorrect,digits=0),"%"),
        paste(round(SVMOutput$PercCorrect,digits=0),"%"),
        paste(round(TreeOutput$PercCorrect,digits=0),"%"),
        paste(round(BaggingOutput$PercCorrect,digits=0),"%"),
        paste(round(BoostingOutput$PercCorrect,digits=0),"%"),
        paste(round(RFOutput$PercCorrect,digits=0),"%"),
        paste(round(NNetOutput$PercCorrect,digits=0),"%")), side = 1, line = 4,
cex = par("cex.lab"), at = c(2,5,8,11,14,17,20,23,26))

mtext("Calc. time", side = 1, line = 3, cex = par("cex.lab"), at = -.8)
mtext("% correct", side = 1, line = 4, cex = par("cex.lab"), at = -.8)



# variable importance------------------------------------------------------------



# the best 2 models after performance evaluation are RF and Boosting
# therefore, this part looks into variable importance of those 2 models

# RF
print(varImp(RF))
plot(varImp(RF), main = "Random Forest - Variable Importance")

# Boosting
print(varImp(Boosting))
plot(varImp(Boosting), main = "Boosting - Variable Importance"
```

Appendix II: SQL Queries

```
-- combining review-checkins table with business and photo table
SELECT t_ruc.business_id
    ,ch_in
    ,date_tip
    ,business_lat
    ,business_long
    ,business_park
    ,business_price
    ,business_open
    ,business_cat
    ,n_photo
    ,cum_n_tips
    ,cum_max_friends
    ,cum_u_names
    ,cum_max_u_elite
    ,cum_max_us_fans
    ,cum_max_us_tip
FROM (
    SELECT t_b.business_id1 AS business_id
        ,business_lat
        ,business_long
        ,business_park
        ,business_price
        ,business_open
        ,business_cat
        ,n_photo
    FROM (
        -- table1: for business
        SELECT business_id AS business_id1
            ,latitude AS business_lat
            ,longitude AS business_long
            ,(
                CASE
                    WHEN STRPOS((attributes::json) ->> 'BusinessParking', 'True')
<> 0
                        THEN 'true'
                    ELSE 'false'
                    END
                ) AS business_park
            ,CAST((attributes::json) ->> 'RestaurantsPriceRange2' AS INTEGER) AS
business_price
            ,is_open AS business_open
            ,categories AS business_cat
        FROM public3.businesstable
        WHERE STRPOS(categories, 'Restaurants') <> 0 AND city='Saint Louis'
        ) AS t_b
    LEFT JOIN (
        -- table2: for photos
        SELECT business_id AS business_id2
```

```
            ,COUNT(*) AS n_photo
        FROM public3.phototable
        GROUP BY business_id
        ) AS t_p ON t_b.business_id1 = t_p.business_id2
    ) AS t_bp
    ,(
    SELECT t_ru.business_id6 AS business_id
            ,(CASE WHEN t_ch.date3 IS NULL THEN 0 ELSE 1 END) AS ch_in
            ,t_ru.date6 AS date_tip
            ,t_ru.cum_n_tips AS cum_n_tips
            ,t_ru.cum_max_friends AS cum_max_friends
            ,t_ru.cum_u_names AS cum_u_names
            ,t_ru.cum_max_u_elite AS cum_max_u_elite
            ,t_ru.cum_max_us_fans AS cum_max_us_fans
            ,t_ru.cum_max_us_tip AS cum_max_us_tip
        FROM (
            -- table3: for checkins
            SELECT t3_1.business_id AS business_id3
                ,date1::DATE AS date3
            FROM (
                SELECT public3.checkintable.business_id AS business_id
                    ,unnest(string_to_array(DATE, ',')) AS date1
                FROM public3.checkintable, public3.businesstable
                WHERE public3.checkintable.business_id =
public3.businesstable.business_id AND public3.businesstable.city='Saint Louis'
                ) AS t3_1
            GROUP BY business_id3
                ,date3
            ) AS t_ch
        RIGHT JOIN (
            -- table6.2: a much more elegant, but more complex query
            SELECT tip_user.business_id51 AS business_id6
                ,tip_user.date5 AS date6
                ,tip_user.n_tips AS n_tips
                ,tip_user.cum_n_tips AS cum_n_tips
                ,(
                    SELECT max(max_us_friends) AS cum_max_friends
                    FROM (
                        SELECT business_id
                            ,DATE
                            ,max(users.n_friends) AS max_us_friends
                        FROM public3.tipstable
                        LEFT JOIN (
                            SELECT user_id AS user_id
                                ,array_length(string_to_array(users.friends,
',',), 1) AS n_friends
                            FROM public3.userstable AS users
                            ) AS users ON public3.tipstable.user_id =
users.user_id
                        GROUP BY business_id
                            ,DATE
                        ) AS t53
```

```
                    WHERE t53.business_id = tip_user.business_id51
                        AND t53.DATE::DATE < tip_user.date5
                    )
                ,(
                    SELECT STRING_AGG(DISTINCT u_names, ',') AS cum_u_names
                    FROM (
                        SELECT business_id
                            ,DATE
                            ,STRING_AGG(DISTINCT users.u_name, ',') AS u_names
                        FROM public3.tipstable
                        LEFT JOIN (
                            SELECT user_id AS user_id
                                ,name AS u_name
                            FROM public3.userstable AS users
                            ) AS users ON public3.tipstable.user_id =
users.user_id
                        GROUP BY business_id
                            ,DATE
                        ) AS t53
                    WHERE t53.business_id = tip_user.business_id51
                        AND t53.DATE::DATE < tip_user.date5
                    )
                ,(
                    SELECT max(max_u_elite) AS cum_max_u_elite
                    FROM (
                        SELECT business_id
                            ,DATE
                            ,max(users.n_elite) AS max_u_elite
                        FROM public3.tipstable
                        LEFT JOIN (
                            SELECT user_id AS user_id
                                ,array_length(string_to_array(users.elite, ','),
1) AS n_elite
                            FROM public3.userstable AS users
                            ) AS users ON public3.tipstable.user_id =
users.user_id
                        GROUP BY business_id
                            ,DATE
                        ) AS t53
                    WHERE t53.business_id = tip_user.business_id51
                        AND t53.DATE::DATE < tip_user.date5
                    )
                ,(
                    SELECT max(max_us_fans) AS cum_max_us_fans
                    FROM (
                        SELECT business_id
                            ,DATE
                            ,max(users.u_fans) AS max_us_fans
                        FROM public3.tipstable
                        LEFT JOIN (
                            SELECT user_id AS user_id
                                ,fans AS u_fans
```

```
                            FROM public3.userstable AS users
                            ) AS users ON public3.tipstable.user_id =
users.user_id
                        GROUP BY business_id
                            ,DATE
                        ) AS t53
                    WHERE t53.business_id = tip_user.business_id51
                        AND t53.DATE::DATE < tip_user.date5
                    )
                ,(
                    SELECT max(max_us_tip) AS cum_max_us_tip
                    FROM (
                        SELECT business_id
                            ,DATE
                            ,max(users.us_tip) AS max_us_tip
                        FROM public3.tipstable
                        LEFT JOIN (
                            SELECT user_id AS user_id
                                ,review_count AS us_tip
                            FROM public3.userstable AS users
                            ) AS users ON public3.tipstable.user_id =
users.user_id
                        GROUP BY business_id
                            ,DATE
                        ) AS t53
                    WHERE t53.business_id = tip_user.business_id51
                        AND t53.DATE::DATE < tip_user.date5
                    )
            FROM (
                SELECT t52.business_id51 AS business_id51
                    ,t52.date5 AS date5
                    ,t52.n_tips AS n_tips
                    ,(
                        SELECT COUNT(t51.TEXT)
                        FROM public3.tipstable AS t51
                        WHERE t51.business_id = t52.business_id51
                            AND t51.DATE::DATE < t52.date5
                        ) AS cum_n_tips
                FROM (
                    SELECT business_id53 AS business_id51, date53 AS date5,
n_tips
                    FROM (SELECT tip.business_id AS business_id53,
date_trunc('day', generate_series
                            ( min(tip.DATE)::timestamp
                            , max(tip.DATE)::timestamp
                            , '1 day'::interval))::date AS date53
                        FROM public3.tipstable AS tip, public3.businesstable AS
bus
                        WHERE tip.business_id=bus.business_id AND bus.city='Saint
Louis'
                        GROUP BY tip.business_id) AS t53
                        LEFT JOIN
```

```
                            (SELECT tip.business_id AS business_id5x
                                                    ,DATE::DATE AS date5x
                                                    ,COUNT(tip.TEXT) AS n_tips
                                                FROM public3.tipstable AS tip,
public3.businesstable AS bus
                                                WHERE
tip.business_id=bus.business_id AND bus.city='Saint Louis'
                                                GROUP BY tip.business_id
                                                    ,date5x) AS t5x
                            ON business_id5x=business_id53 AND date5x=date53
                    ) AS t52
                ) AS tip_user
            ) AS t_ru ON t_ch.date3 = t_ru.date6
            AND t_ch.business_id3 = t_ru.business_id6
        WHERE cum_n_tips <> 0
    ) AS t_ruc
WHERE t_bp.business_id = t_ruc.business_id;
```

Goethe-Universität | 60629 Frankfurt am Main
Fachbereich Wirtschaftswissenschaften | Prüfungsamt
(Stand: 04.06.2018)

GOETHE UNIVERSITÄT FRANKFURT AM MAIN

Ehrenwörtliche Erklärung/ Statutory Declaration

An das Ende von schriftlichen Arbeiten (bspw. Seminararbeiten, Bachelor- oder Masterarbeiten) ist eine ehrenwörtliche Erklärung zu setzen. Sie hat folgenden Wortlaut:

Ehrenwörtliche Erklärung

"Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Wörtlich übernommene Sätze oder Satzteile sind als Zitat belegt, andere Anlehnungen, hinsichtlich Aussage und Umfang, unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist nicht veröffentlicht. Sie wurde nicht, auch nicht auszugsweise, für eine andere Prüfungs- oder Studienleistung verwendet."

Ort, Datum: Frankfurt, 17.07.2025 Unterschrift: ……..…….

Bei schriftlichen Arbeiten, die zusätzlich in elektronischer Form einzureichen sind, ist jedoch folgender Wortlaut zu verwenden:

Ehrenwörtliche Erklärung

"Ich versichere hiermit, dass ich die vorliegende Arbeit selbständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel verfasst habe. Wörtlich übernommene Sätze oder Satzteile sind als Zitat belegt, andere Anlehnungen, hinsichtlich Aussage und Umfang, unter Quellenangabe kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen und ist nicht veröffentlicht. Sie wurde nicht, auch nicht auszugsweise, für eine andere Prüfungs- oder Studienleistung verwendet. Zudem versichere ich, dass die von mir abgegebenen schriftlichen (gebundenen) Versionen der vorliegenden Arbeit mit der abgegebenen elektronischen Version auf einem Datenträger inhaltlich übereinstimmen."

Ort, Datum: ……..……. Unterschrift: ……..…….

A statutory declaration is to be included at the end of every written work (e.g. Seminar paper, Bachelor's or Master's thesis).
The translation is as follows:

Statutory Declaration

"I herewith declare that I have composed the present thesis myself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance."

Place, Date: Frankfurt, 17.07.2025 Signature: ……..…….

In case of written work, which is also to submit on a data carrier, the translation is as follows:

Statutory Declaration

"I herewith declare that I have composed the present thesis myself and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The thesis in the same or similar form has not been submitted to any examination body and has not been published. This thesis was not yet, even in part, used in another examination or as a course performance. Furthermore I declare that the submitted written (bound) copies of the present thesis and the version submitted on a data carrier are consistent with each other in contents."

Place, Date: ……..……. Signature: ……..…….

AACSB ACCREDITED

Turnitin Receipt

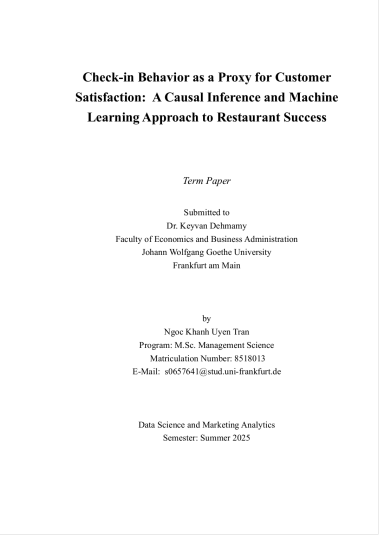
turnitin

Digital Receipt

This receipt acknowledges that Turnitin received your paper. Below you will find the receipt information regarding your submission.

The first page of your submissions is displayed below.

Submission author:	Ngoc Khanh Uyen Tran
Assignment title:	Term Paper
Submission title:	TRAN, Ngoc Khanh Uyen (Student ID: 8518013)
File name:	TRAN_Ngoc-Khanh-Uyen_8518013_Paper.pdf
File size:	1.27M
Page count:	38
Word count:	10,650
Character count:	66,988
Submission date:	17-Jul-2025 02:59PM (UTC+0200)
Submission ID:	2715642531



Check-in Behavior as a Proxy for Customer Satisfaction: A Causal Inference and Machine Learning Approach to Restaurant Success

Term Paper

Submitted to
Dr. Keyvan Dehmamy
Faculty of Economics and Business Administration
Johann Wolfgang Goethe University
Frankfurt am Main

by
Ngoc Khanh Uyen Tran
Program: M.Sc. Management Science
Matriculation Number: 8518013
E-Mail: s0657641@stud.uni-frankfurt.de

Data Science and Marketing Analytics
Semester: Summer 2025