airbnb

Guest Satisfaction Prediction

TEAM SC_3

نوران احمد سمير احمد	2022170472
محمد خالد جمال عرابي	2022170365
رانا ناصر محمد جودة	2022170145
عبدالرحمن احمد طاهر	2022170227
سلمي عماد احمد علي	2022170189
شهد سامح عبد العزيز	2022170206

Introduction

"Our Guest Satisfaction prediction project advanced machine learning techniques to analyze and predict guest satisfaction levels for Airbnb stays. By examining features such as property characteristics, host behavior, and guest feedback, our model delivers accurate and insightful predictions. Focused on enhancing the guest experience and supporting hosts in improving their services, our work empowers the Airbnb community with data-driven insights. Join us in our mission to create more enjoyable, personalized, and memorable travel experiences for guests around the world."

Objective

Build a regression model to accurately predict acquisition prices
Identify key predictors, including founder/board member attributes Compare multiple regression algorithms and their performance

Scope

The project implements supervised regression on datasets, focusing on preprocessing, feature engineering, and model comparison. The algorithms include Linear Regression, Polynomial Regression, Random Forest, XGBoost, and SVR, lasso , ridge, DT , RF ,Elastic, Hyper, Gradiant Boosting, Log tranform

1. Preprocessing Techniques

The provided dataset contained various types of features, including numerical, categorical, and textual data. To ensure consistency and prepare the data for modeling, the following preprocessing steps were performed:

1.1 Handling Missing Values and Duplicates

- **Numerical Features:** Missing values in numerical features such as reviews_per_month were imputed using the KNN imputer
- **Categorical Features:** Features such as host_response_rate and host_is_superhost had missing values filled with the mode (most frequent value).
- For features with too many missing values (>70% missing), such as thumbnail_url, we **dropped** the column because imputing would have introduced too much noise.
- Duplicates dropped
- Drop host_listings_count after calculating the match between it and host_total_listings_count, and it was above 99%

1.2 Encoding Categorical Variables

- **Technique:**
    - **One-Hot Encoding** was applied to nominal categories (e.g., bed_type).
    - **Label Encoding** was used for ordinal features if ordering made sense (e.g., amenities).
    - **Target Encoding**
    -

1.3 Outlier Detection and Removal

- **Technique:**

Outliers in fields like price, number_of_reviews were detected via boxplots and removed using the IQR method.

2. Feature Analysis

We explored feature relationships using correlation matrices, scatter plots, and pair plots.

Key Observations:

- **Overall, most features show a very weak correlation with review_scores_rating.**
- No feature has a moderate (>|0.3|) or strong correlation.
- number_of_reviews and number_of_stays show a small positive correlation (~0.066) with review_scores_rating.
- **Price-related features** (nightly_price, price_per_stay) have **extremely weak** positive correlation (~0.01 to 0.02).
- **Property size features** (like bedrooms, bathrooms, accommodates, beds) show strong mutual correlations (0.6-0.8) but have **very weak** direct **correlation** to review_scores_rating.
- **Location features** (latitude, longitude) have **negligible correlation** with the target.

3. Regression Techniques Used

We implemented 15 very different regression models to predict guest satisfaction:

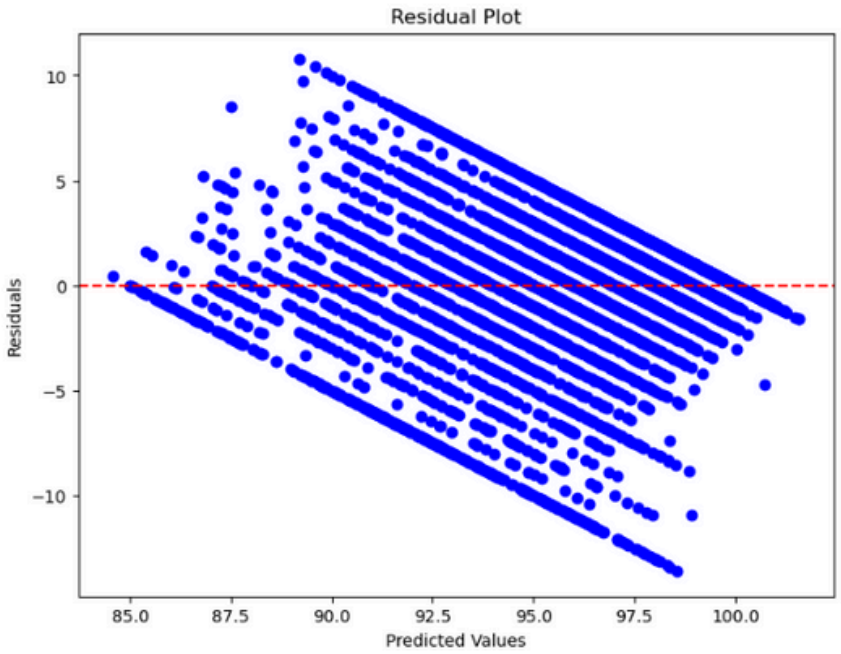
3.1) Linear Regression

Linear regression is a foundational machine learning algorithm used to predict a target variable based on the linear relationship between input features and the target. The model assumes that the relationship between the independent variables (features) and the dependent variable (target) is linear.

Key Metrics:

- **R^2 (R-squared)**: 0.4292 – Represents the proportion of the variance in the target variable that is explained by the model. The R^2 score indicates that about 42.92% of the variance is explained by the model.
- **MAE (Mean Absolute Error)**: 2.4394 – Indicates the average absolute difference between the predicted and actual values.
- **MSE (Mean Squared Error):** 11.7083 – Measures the average squared difference between predicted and actual values.
- **RMSE (Root Mean Squared Error):** 3.4217 – The square root of MSE, giving a sense of the magnitude of errors in the same units as the target variable.

.



3.2) Huber Regression

Huber regression is a robust regression model that combines the properties of linear regression and absolute error minimization, making it less sensitive to outliers. It uses a combination of squared error for smaller residuals and absolute error for larger residuals.

Key Metrics:

- **R^2**: 0.3934 – A slightly lower R^2 than linear regression, suggesting the model explains less variance in the target variable.
- **MAE:** 2.3699 – Slightly better MAE compared to linear regression, indicating less deviation between predicted and actual values.
- **RMSE**: 3.4924 – Higher RMSE compared to Linear Regression, suggesting that despite being robust to outliers, the model's overall error is higher.

.

3.3) ElasticNet Regression

ElasticNet is a linear regression model that combines the penalties of both **Lasso (L1 regularization)** and **Ridge (L2 regularization)**. It is particularly useful when there are many correlated features, as it can both shrink and select features.

Key Metrics:

- **R^2**: 0.4183 – Slightly better than both Linear and Huber regression models, indicating that the model can explain about 41.83% of the variance in the data.
- **MAE**: 2.4361 – Very similar to Linear Regression's MAE
- **RMSE**: 3.4206 – Close to the Linear Regression's RMSE

3.4) Ridge Regression

Ridge regression is a type of **linear regression** that applies **L2 regularization** (penalty on the size of coefficients), which helps to prevent overfitting by shrinking large coefficients. This model is useful when the dataset has many features.

Key Metrics:

- **R^2**: 0.4187 – Very similar to ElasticNet and slightly better than Linear Regression.
- **MAE:** 2.4380 – Similar to Linear and ElasticNet models.
- **RMSE:** 3.4187 – Matches the RMSE of ElasticNet, indicating comparable predictive performance.

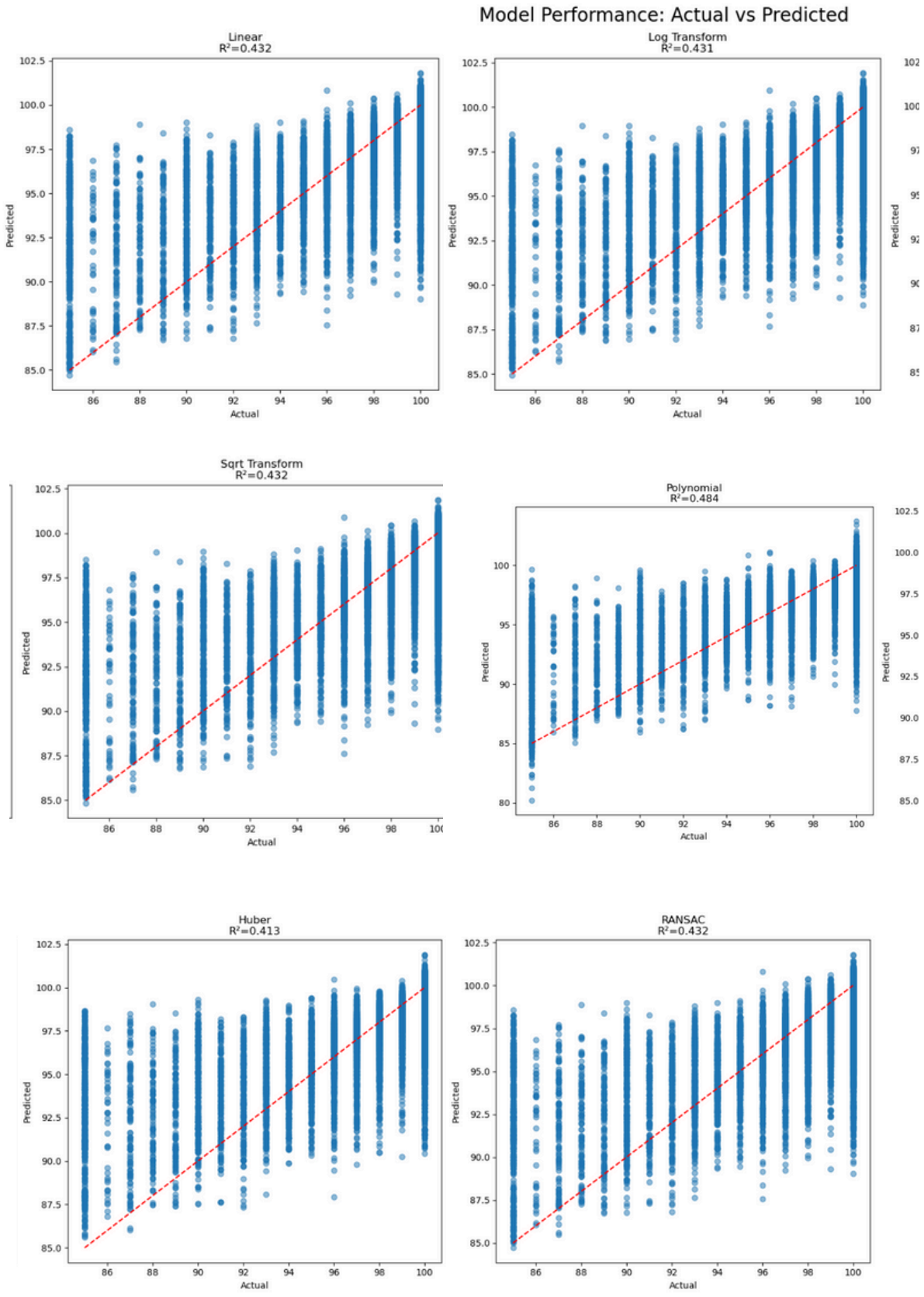
3.5) Lasso Regression

Lasso regression is another form of **regularized linear regression** that applies **L1 regularization**, which can drive some coefficients to zero. This makes Lasso particularly useful for feature selection.

Key Metrics:

- **R^2**: 0.4154 – Similar to Ridge and ElasticNet, with a small decrease from Linear Regression.
- **MAE:** 2.4450 – Slightly worse than Ridge and ElasticNet.
- **RMSE**: 3.4284 – Slightly higher than both Ridge and ElasticNet.

Plots

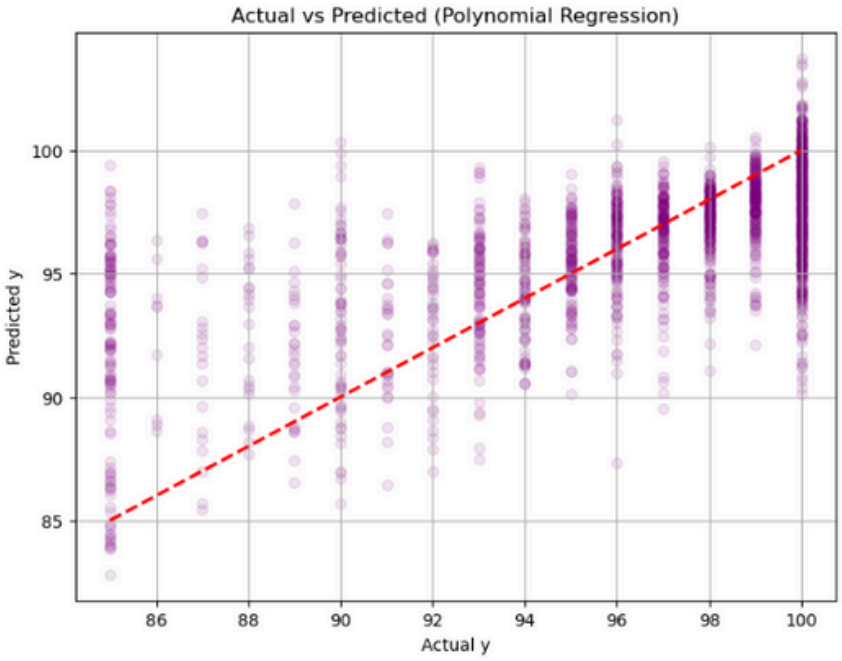

Model Performance: Actual vs Predicted
Linear
R²=0.432
Log Transform
R²=0.431
Sqrt Transform
R²=0.432
Polynomial
R²=0.484
Huber
R²=0.413
RANSAC
R²=0.432
Predicted
Actual

3.6) Polynomial Regression

Polynomial regression is an extension of linear regression that allows for non-linear relationships between the independent variables and the target variable. It creates polynomial features based on the original features.

Key Metrics:

- **R^2**: **0.4841 – The best R^2 score among all the models**, indicating that it explains more variance in the data than the other models.
- **MAE**: 2.2404 – Better than most models, indicating smaller errors on average.
- **RMSE**: 3.1765 – The lowest RMSE, suggesting that the predictions are more accurate.



3.7) Random Forest Regression

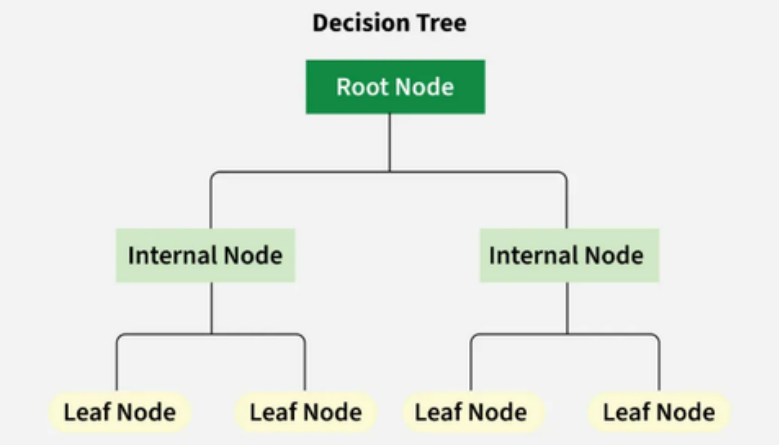
Random Forest is an ensemble learning method that creates multiple decision trees during training and outputs the mean prediction of the individual trees. It is very powerful for capturing non-linear relationships and interactions between features.

Key Metrics:

- **R^2**: 0.3862 – Slightly lower R^2 compared to polynomial regression, suggesting that this model explains less of the variance.
- **MAE**: 2.3402 – Similar to Linear Regression and other models.
- **MSE**: 12.3430 – A higher MSE compared to most linear models, which suggests more variance in prediction errors.

3.8) Decision Tree

Decision Trees are the foundation for many classical machine learning algorithms like Random Forests, Bagging, and Boosted Decision Trees. His idea was to represent data as a tree where each internal node denotes a test on an attribute (basically a condition), each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label.



Why Use Decision Trees?

- Decision Trees usually mimic human thinking ability while making a decision, so it is easy to understand.
- The logic behind the decision tree can be easily understood because it shows a tree-like structure.

How to achieve a great result?

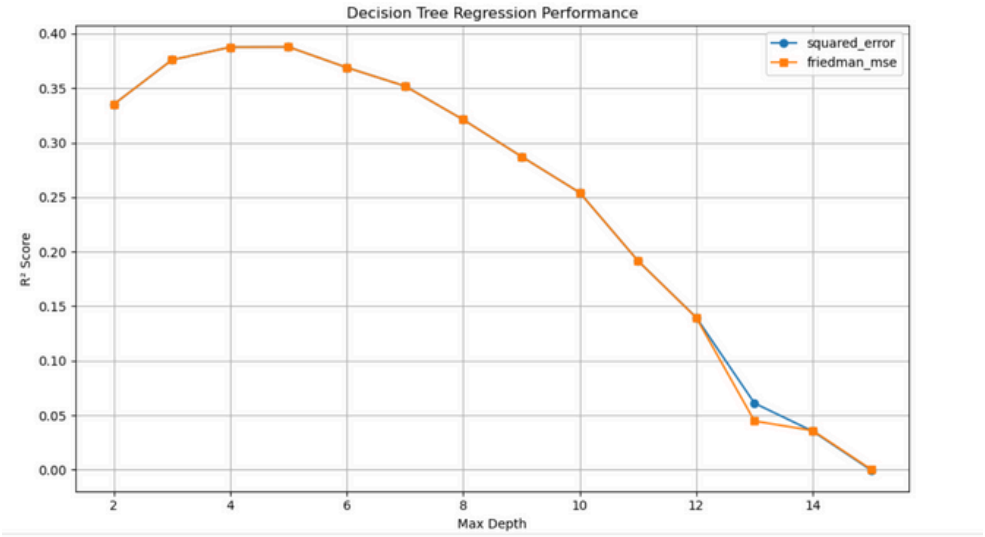
1- Gridsearch is used to compare between hyperparameters to get the best ones after plotting the results we noted that:

- The best max depth is 4.
- Mean Squared Error (MSE): 12.2953
- Mean Absolute Error (MAE): 2.3996
- R^2 Score: 0.3885

2- **Comparing between two splitting criteria:**

- Squared_error
- Friedman_mse

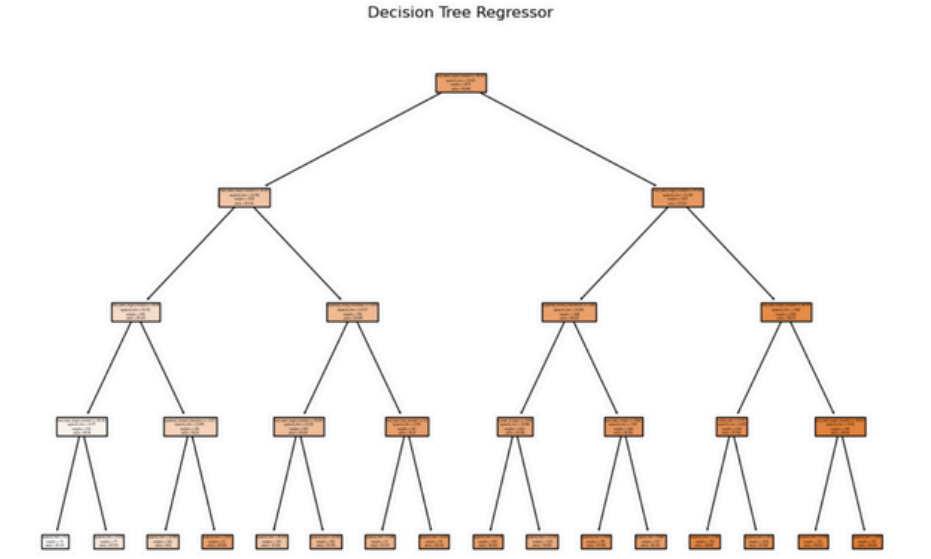
Comparing between two splitting criteria:



Then training the DT on the suitable parameters, according to r^2 scoring making cross validation = 10 samples/set

Final result:

- Decision_Tree R2_Train: 0.4287
- Decision_Tree R2_Test: 0.3876
- CV R^2 Score: 0.4292



3.9) Support Vector Regression

Support Vector Regression (SVR) is an algorithm based on the principles of Support Vector Machines, but designed for regression tasks. It aims to find an approximate function that stays as close as possible to most data points, allowing a certain margin of error (ε). The model seeks to remain as simple as possible and relies only on the points outside this margin (called Support Vectors) to define the final function.

In this work, we developed a Support Vector Regression (SVR) model using the **rbf kernel** to predict target values accurately.

Evaluation:

- Test R^2 Score: 0.3203
- CV R^2 Score: 0.3221

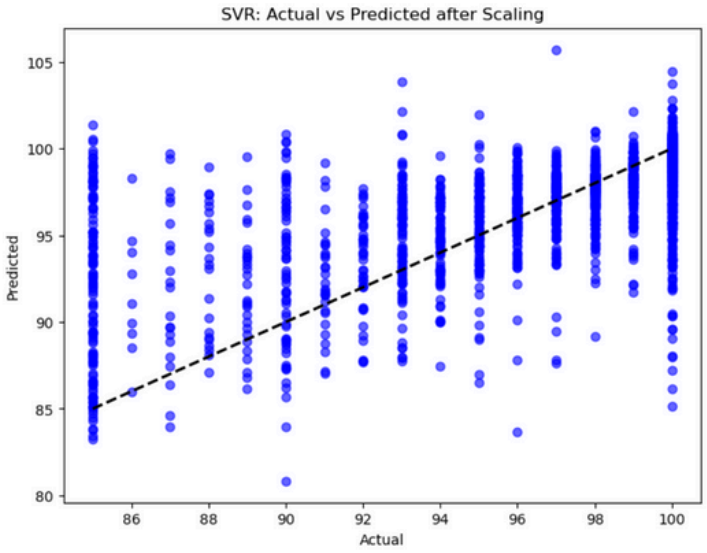
Steps to achieve a great results:

Feature scaling was applied using StandardScaler to standardize the data, ensuring optimal SVR performance and mitigating the effects of different feature scales.

Initialization with optimized hyperparameters (C=100, gamma='scale', epsilon=0.01) to balance model flexibility and robustness against outliers.

10-fold Cross-Validation was conducted, confirming the model's stability and ability to generalize to unseen data.

Actual vs predicted values showing a strong correlation and visually validating the model's effectiveness in capturing the underlying patterns of the data.



3.10) Comparison between Gradient Boosting, Random Forest, XGBoost

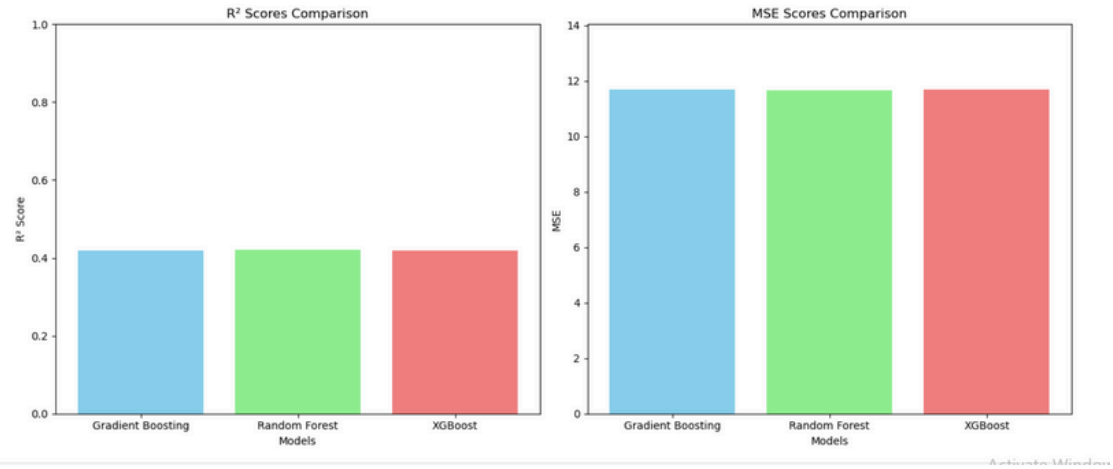
Gradient Boosting is a ensemble learning method used for classification and regression tasks. It is a boosting algorithm which combine multiple weak learner to create a strong predictive model. It works by sequentially training models where each new model tries to correct the errors made by its predecessor.

Random Forest algorithm is a powerful tree learning technique in Machine Learning to make predictions and then we do voting of all the tress to make prediction. They are widely used for classification and regression task

XGBoost is a powerful approach for building supervised regression models. The validity of this statement can be inferred by knowing about its (XGBoost) objective function and base learners. The objective function contains loss function and a regularization term. It tells about the difference between actual values and predicted values

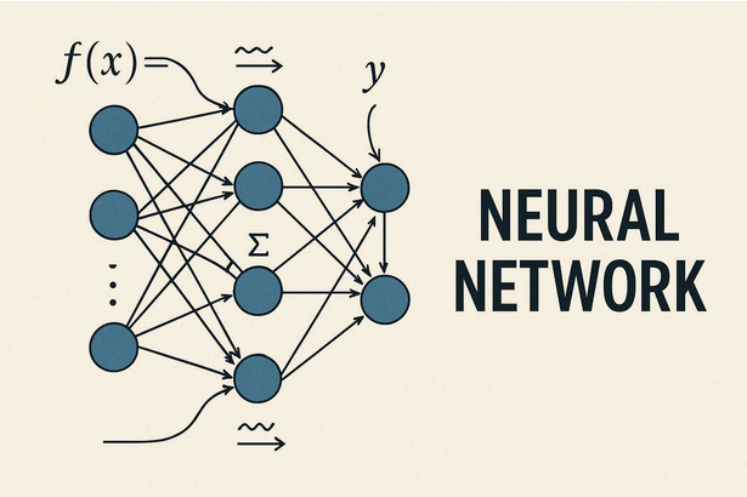
Results according to R^2:

- Gradient Boosting: 0.41806620
- RandomForest: 0.42002479
- XGBoost: 0.41918159



3.11) Neural Network

neural network is a computational model inspired by the way the human brain processes information. It consists of layers of interconnected nodes, or "neurons," that work together to recognize patterns, classify data, and make predictions. Each neuron takes input, processes it using mathematical functions, and passes the result to the next layer. Neural networks are widely used in machine learning tasks like image recognition, natural language processing, and predictive analytics, enabling systems to learn from data and improve over time.



In this work, we developed a Neural Network (NN) Regression Model using Keras and TensorFlow. The model was designed to predict continuous target variables based on input features.

We began by scaling the data using the Min-Max Scaler, ensuring the features were within a normalized range of [0, 1], which helps the neural network learn efficiently. The model itself consisted of an input layer, two hidden layers (with ReLU activation), a dropout layer to mitigate overfitting, and a final output layer with one neuron to predict the target value.

For training, we used 10-fold cross-validation to evaluate the model's performance on different data splits. We also computed R^2, RMSE, and MAE to assess the model's predictive accuracy.

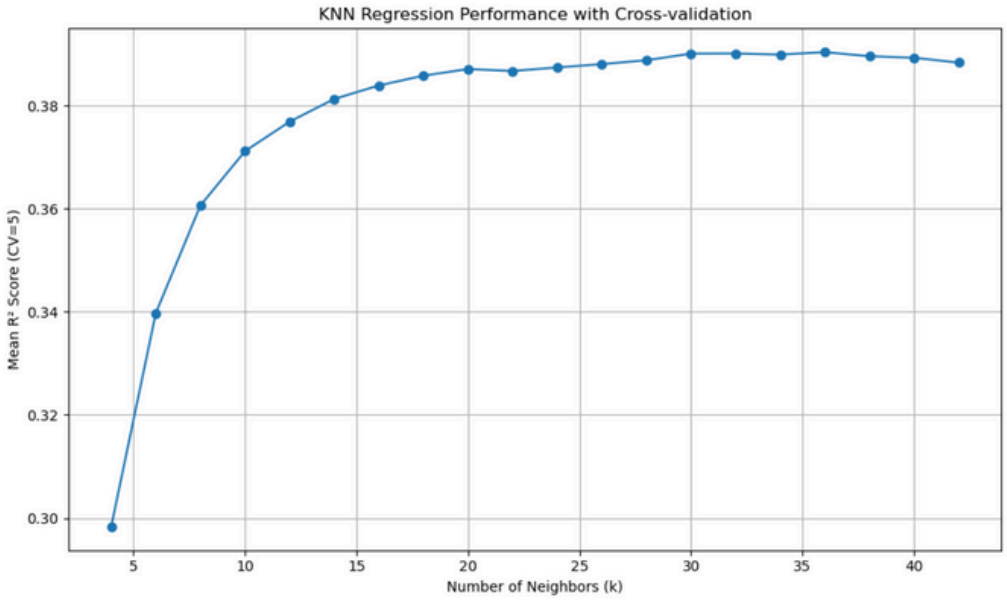
Results of Neural Network

- R2: 0.3857
- RMSE: 3.5145
- MAE: 2.5387
- CV R2: 0.3787 ± 0.0674

3.12) KNN

KNN regression is a non-parametric method used for predicting continuous values. The core idea is to predict the target value for a new data point by averaging the target values of the K nearest neighbors in the feature space.

To get the current result:
K is changed in range to get best r2
and cross validation technique is applied too!



Best number of neighbors (k): 36 with $R^2 = 0.3904$

3.13 & 3.14) CatBoost & LightGBM

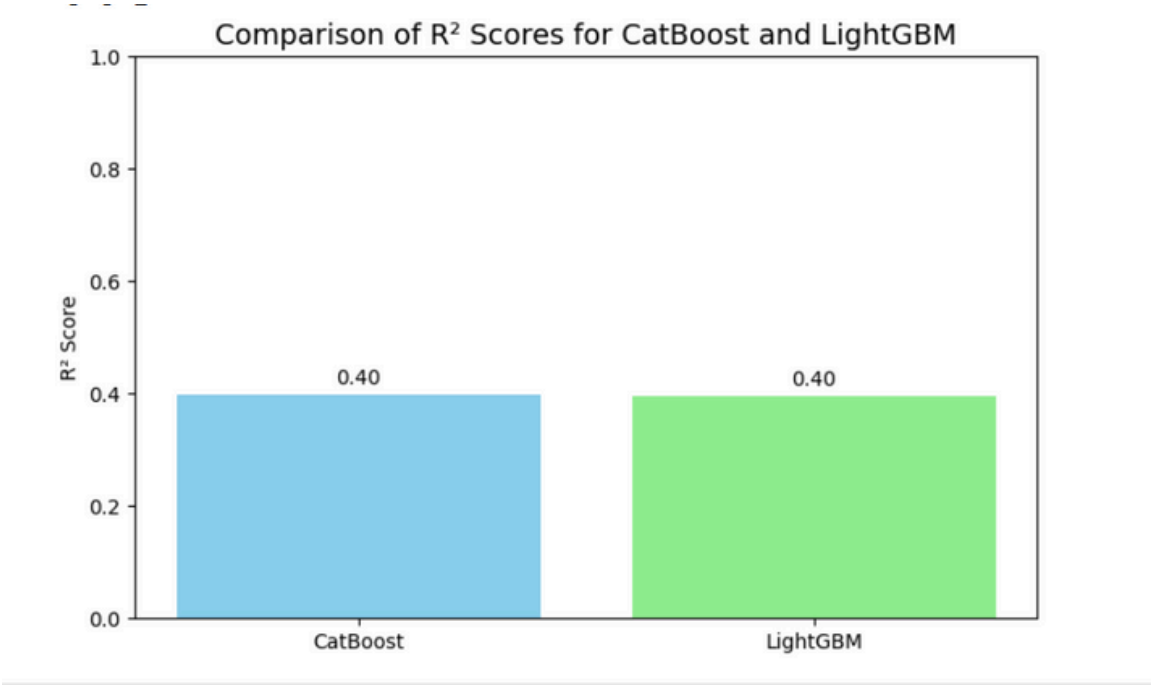
LightGBM and CatBoost are both powerful gradient boosting frameworks designed for efficient, high-performance machine learning.

LightGBM (Light Gradient Boosting Machine) is known for its speed and scalability, particularly in large datasets. It uses histogram-based methods for faster training and reduced memory usage, making it a top choice for many machine learning tasks.

On the other hand, CatBoost (Categorical Boosting) is optimized for handling categorical features without the need for extensive preprocessing like one-hot encoding. It's particularly useful when working with datasets that contain categorical data and offers strong performance with minimal tuning. Both frameworks are popular for classification and regression tasks, offering robust handling of complex data with high accuracy.

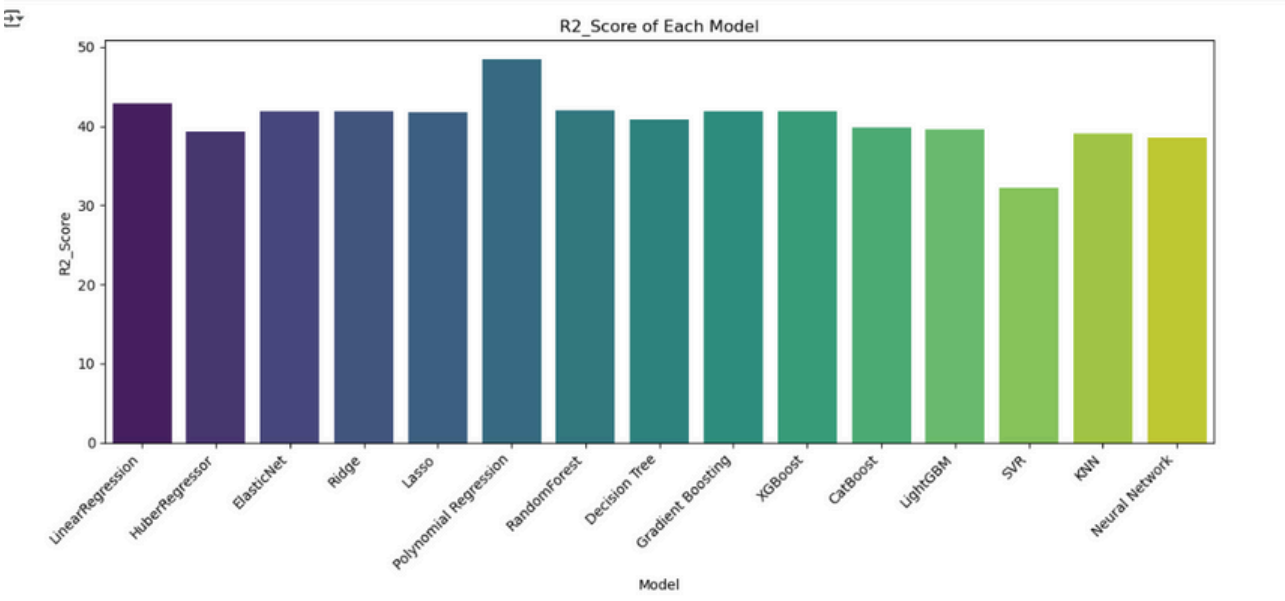
Results:

- catboost_model R^2: 0.3988
- lightgbm_model R^2: 0.3961



R^2 according different Models

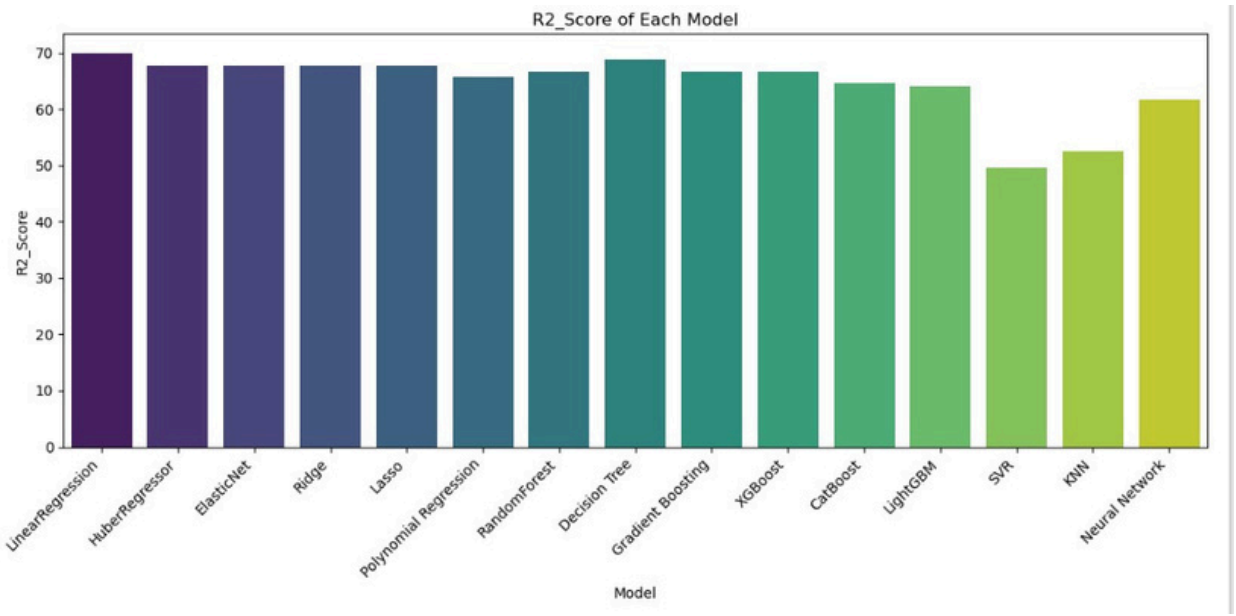
	Model	R2_Score
5	Polynomial Regression	48.421344
0	LinearRegression	42.915668
6	RandomForest	42.002479
9	XGBoost	41.918159
2	ElasticNet	41.834022
8	Gradient Boosting	41.806620
3	Ridge	41.805614
4	Lasso	41.787977
7	Decision Tree	40.873400
10	CatBoost	39.884824
11	LightGBM	39.612626
1	HuberRegressor	39.340759
13	KNN	39.039908
14	Neural Network	38.573110
12	SVR	32.207571



when we use target encoding with train and cross validation and create "host_avg_rate" that gropby "host_id" and "review score rating" mean

[117]:

	Model	R2_Score
0	LinearRegression	69.841522
7	Decision Tree	68.901394
4	Lasso	67.705985
2	ElasticNet	67.694141
3	Ridge	67.693362
1	HuberRegressor	67.652210
8	Gradient Boosting	66.640602
9	XGBoost	66.633419
6	RandomForest	66.562054
5	Polynomial Regression	65.781513
10	CatBoost	64.592223
11	LightGBM	64.066358
14	Neural Network	61.605113
13	KNN	52.523902
12	SVR	49.587542



Summary of Differences between Best Models:

- **Linear Models (Linear, Ridge, Lasso, ElasticNet):**
  - These models generally perform similarly with R^2 scores between 0.39 to 0.42.
  - The regularized versions (Ridge, Lasso, ElasticNet) help to prevent overfitting compared to standard linear regression.
  - Lasso performs feature selection, while Ridge shrinks coefficients to reduce model complexity.

- **Huber Regression:**
  - Slightly lower performance than linear models in terms of R^2 but is more robust to outliers.

- **Polynomial Regression:**
  - Significantly outperforms all other models with the highest R^2 and lowest MAE and RMSE..

- **Random Forest:**
  - More complex and captures non-linear relationships, but at the cost of interpretability. Its performance is comparable to linear models in terms of error metrics but has a slightly lower R^2.

Conclusion

- **Best Model for Predictive Accuracy: Polynomial Regression**, as it provides the highest R^2 score and the lowest error metrics.
- **Best for Robustness to Outliers: Huber Regression** is the most robust to outliers.
- **Best for Simplicity and Regularization: Ridge or ElasticNet** offer a good balance of regularization with simplicity.

4.Features Used and Discarded

In constructing our regression models, a meticulous feature selection and engineering process was executed to ensure both relevance and clarity. Below are the details of features retained and those removed:

• **Features Used:**

- **Numerical Metrics**: Key price variables (nightly price, price per stay), review counts, and host statistics (total listings, average response rate).
- **Categorical Indicators:** Binary flags (superhost status, instant booking), one-hot encoded room and property types, and clustered location labels.
- **Engineered Interactions**: Host experience × superhost status, reviews per stay, price per guest, and other ratio-based features.
- **Text-Derived Insights**: Sentiment compound scores extracted from listing summaries, host bios, and other description fields.
- **Temporal Features:** Days since host registration, first and last review measured from a fixed reference date, including cyclical month representations.
- **Amenities Profile**: Counts of amenities, presence indicators for high-impact amenities (e.g., Wi-Fi, kitchen), and an amenities rarity score based on inverse listing frequency.

• **Features Discarded:**

- **Columns with Excessive Missing Data**: Variables such as host acceptance rate, square footage, and thumbnail URLs were excluded due to over 40% missing values.
- **High-Cardinality Identifiers:** URL and ID fields (listing URL, host ID), street addresses, zipcode, and other location descriptors were removed to simplify the model.
- **Redundant Metrics:** Duplicated host listing counts were consolidated, and raw text fields dropped after sentiment extraction.
- **Low-Variance Binaries**: Flags with near-constant values (e.g., license requirements) were excluded to avoid noise.

5. Dataset Splits

- **Training Set: 80% of the data.**
  - After the train-test split (train_test_split with test_size=0.2), the training set comprises 80% of the total dataset.
  - Size: X_train.shape (**Training set size: (6979, 30)**).
- **Testing Set: 20% of the data.**
  - The remaining 20% is reserved for testing.
  - Size: X_test.shape (**Test set size: (1745, 30)**).
- **Validation Set:** Not explicitly split as a separate set.
  - Instead, **cross-validation** is used within the training set for model evaluation and hyperparameter tuning (e.g., cv=10 in GridSearchCV and cross_val_score). This effectively uses subsets of the training data as validation sets during the process.

The exact sizes depend on the original dataset (GuestSatisfactionPrediction.csv), but the proportions are fixed at 80% training and 20% testing, with validation handled via cross-validation.

6. Techniques to Improve Model Performance

A suite of preprocessing and modeling strategies was employed to maximize predictive accuracy and generalization capability:

1. Data Preprocessing:

- **Imputation:** KNN imputation for numeric gaps; mode substitution for categorical nulls.
- **Outlier Treatmen**t: Winsorization via IQR-based capping to mitigate extreme values.
- **Transformation**: Logarithmic scaling of skewed features (review counts, price variables).

2. Feature Engineering:

- **Interaction Terms:** Combined host and listing attributes (e.g., superhost_review_interaction).
- **Clustering:** KMeans clustering for host behavior and geographic coordinates to reveal latent groupings.
- **Text Sentiment**: NLP sentiment analysis to derive numeric sentiment features from listing descriptions.

3. Feature Selection:

- **Mutual Information:** SelectKBest with mutual_info_regression to identify the top 30 predictors.
- **Correlation Filtering:** Removal of features with correlation >0.9 to reduce multicollinearity.

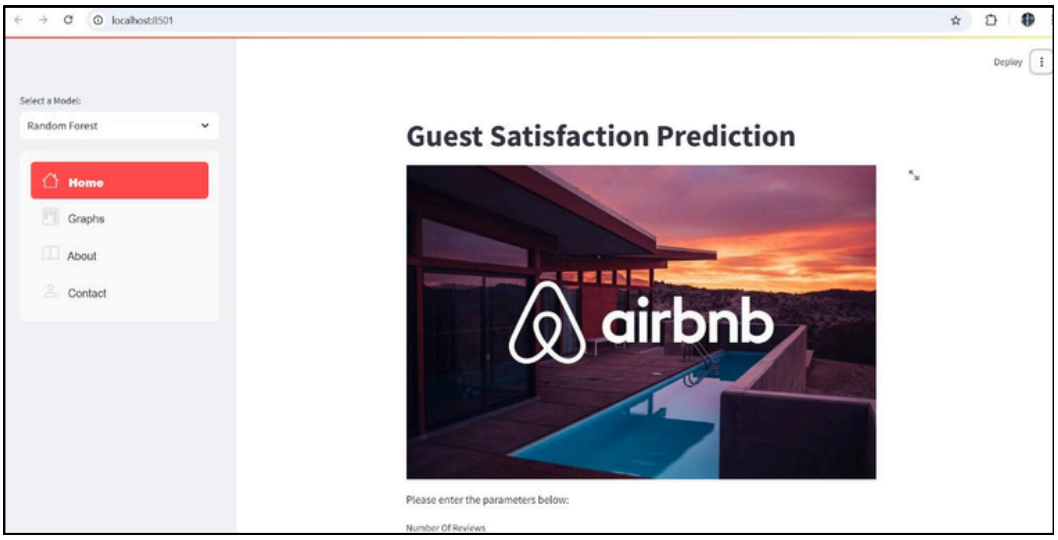
4. Scaling and Encoding:

- **StandardScaler**: Applied to normalize numeric features before model fitting.
- **One-Hot & Target Encoding:** Categorical handling based on cardinality and predictive relevance.

5. Model Tuning and Selection:

- **Cross-Validation**: 10-fold CV used throughout GridSearchCV for ridge, lasso, tree-based, and boosting models.
- **Ensemble Strategies**: Voting and stacking regressors combining Random Forest, XGBoost, and LightGBM.
- **Residual Correction**: Secondary tree-based model trained on linear regression residuals for improved fit.
- **Advanced Models**: Neural network architecture with two hidden layers and dropout to capture complex patterns.

GUI



This project is a Guest Satisfaction Prediction Web Application developed using Streamlit.
It machine learning models (Linear Regression, Decision Tree, Random Forest) to predict guest satisfaction levels for Airbnb stays based on multiple input features such as number of reviews, host listings count, minimum nights, price, and more.

Key features include:

Interactive Web Interface built with Streamlit.

User Input Fields for property and guest details.

Model Selection from the sidebar to choose the prediction algorithm.

Graphs and Visualizations showcasing relationships between features.

About Page describing the project goals.

Contact Form allowing users to send feedback directly to the developer's email.

Additional technologies:

Web animations using Lottie Files.

SMTP email integration for user feedback.

Data preprocessing and model training with Scikit-learn.

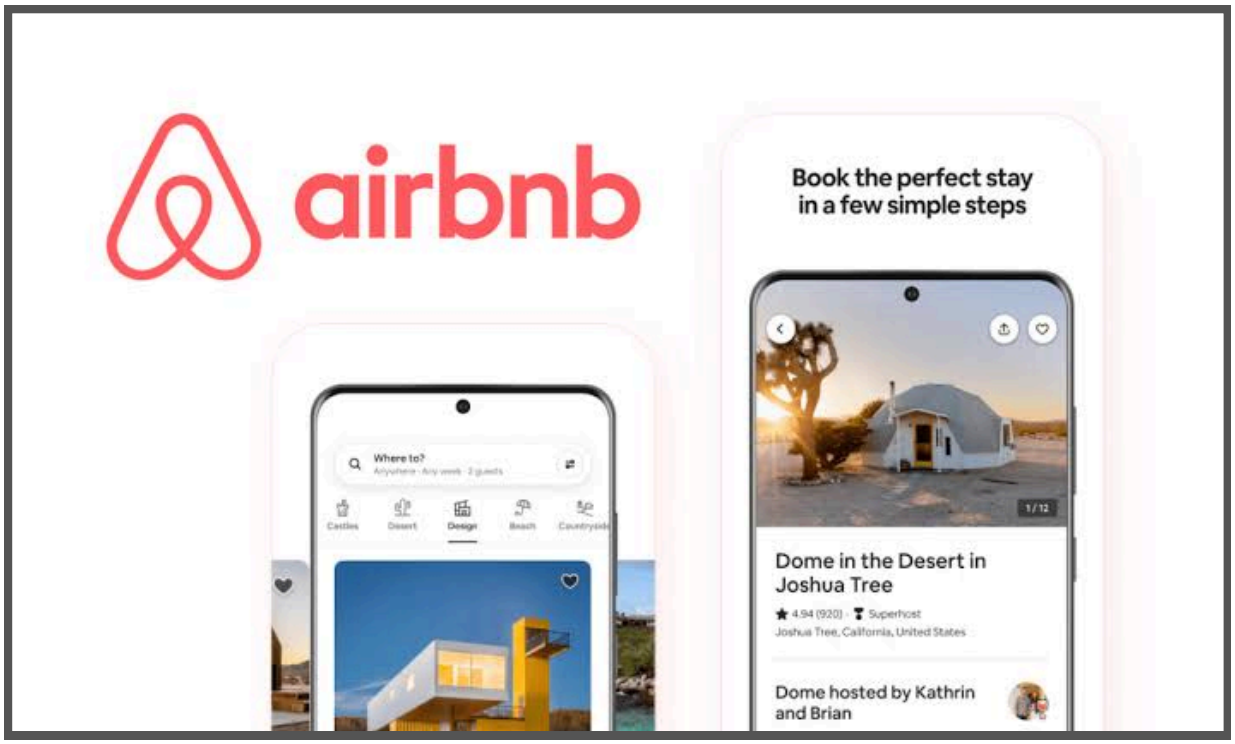
The project aims to empower hosts and Airbnb users by providing accurate satisfaction predictions to enhance the guest experience.

Mention any further techniques that were used to improve the results

We use Feature Engineering to increase corr with the target
AND
Web Scraping, but it needs LogIn:
This script scrapes user profile data from Airbnb after logging in automatically.
Since Airbnb sometimes requires 2FA verification (OTP sent via email), the script also handles fetching the OTP from Gmail automatically.
It uses Selenium (via undetected_chromedriver to avoid bot detection), BeautifulSoup to parse HTML, and IMAP to read emails.

Conclusion

In conclusion, our project combines the power of data-driven web scraping and machine learning to enhance guest satisfaction within the Airbnb community. By securely logging in and extracting valuable user profile data, we lay the foundation for building predictive models that accurately assess guest experiences. The insights gained from analyzing property details, host interactions, and guest feedback enable both guests and hosts to make more informed decisions. Through this work, we aim to foster a more personalized, satisfying, and memorable travel experience for all Airbnb users, advancing the future of hospitality with technology and innovation.



Thank You!

Project Report

Models && Preprocessing

Project Name:	Team : SC_3	Type: Classification
Guest satisfaction		

Part 2

Rana Nasser Mohamed	2022170145
AbdElRahman Ahmed Taher	2022170227
Nouran Ahmed Samir	2022170472
Shahd Sameh AbdElAziz	2022170206
Mohamed Khalid Gamal	2022170365
Salma Emad Ahmed	2022170189

Objective

This report details the classification phase of the machine learning project focused on predicting guest satisfaction for Airbnb listings. The primary objective of this milestone (Milestone 2) was to classify guest satisfaction levels into one of three categories: Average, High, or Very High, using a cleaned and feature-engineered dataset. A range of classification models were evaluated, including Decision Tree, Random Forest, and Logistic Regression, with performance assessed using tools such as scikit-learn, cross-validation, and GridSearchCV for hyperparameter tuning. This report summarizes the classification accuracy, training and testing times, explains the feature selection and preprocessing pipeline.

preprocessing

Target Variable Processing:

For the classification task, we transformed the categorical guest satisfaction column into an ordinal numerical feature:

Mapping:

Average → 0

High → 1

Very High → 2

Handling Missing Values:

Removed or imputed rows with missing satisfaction values to ensure reliable target labels

Other Features:

1. All other features were preprocessed similarly to Milestone 1 (Regression task), including: Handling missing values using strategies such as mean, median, or mode imputation.
2. Encoding categorical variables using frequency or ordinal encoding depending on context.
3. Feature engineering, including:

-
- Aggregating host and property details (e.g., total listings, price per stay)
- Converting Boolean and binary columns to 0/1.
- Normalization and scaling of numerical features where appropriate.
- Sampling by using **SMOTE**

Key Differences from Regression:

Target Variable:

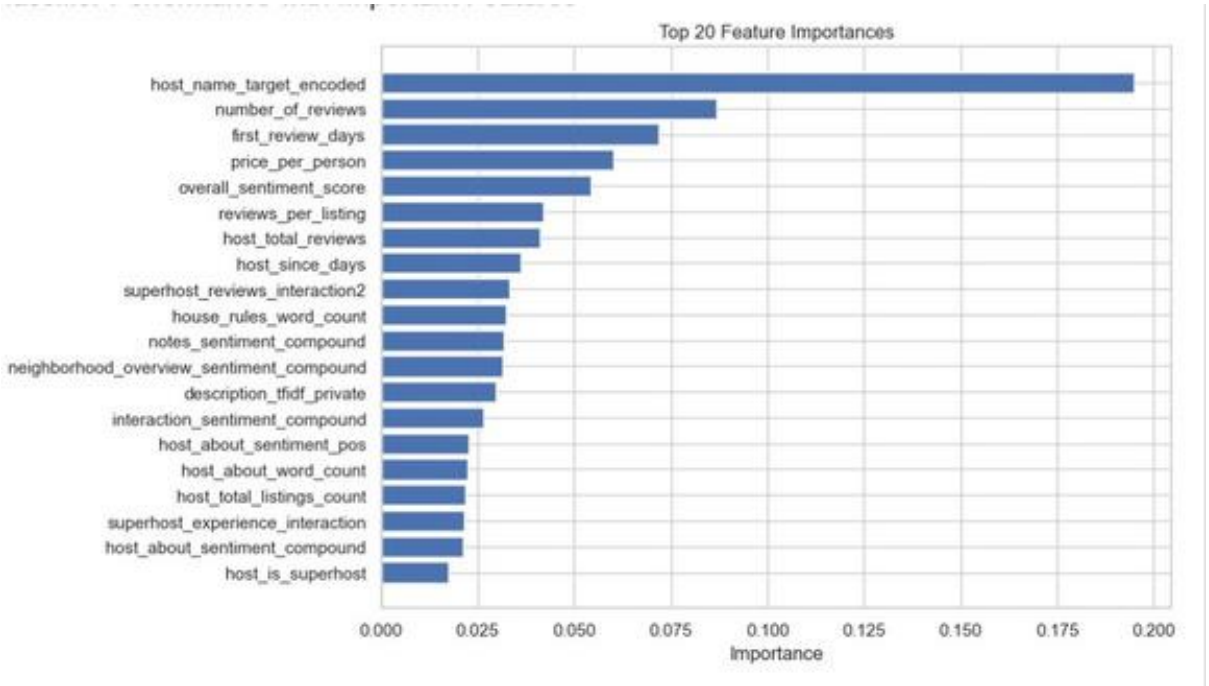
The Regression task used numerical ratings (review_scores_rating), while the Classification task grouped satisfaction into ordinal categories (guest_satisfaction).

Goals

- Predict discrete satisfaction levels rather than a continuous rating score.

Feature Selection:

The feature selection process for this classification task was essential to identify the most relevant attributes for predicting guest satisfaction levels (Average, High, Very High). The approach involved evaluating both categorical and numerical features using appropriate statistical and machine learning methods, such as correlation analysis for numerical variables and chi-square tests for categorical variables. The target variable, guest satisfaction, represents the three satisfaction categories used in the classification.



Feature Selection for GuestSatisfactionPredictionMilestone2:

The feature selection process in this milestone utilized a hybrid NoVAA strategy (No Variance, Anti-correlation, and Attribute ranking) to identify the most relevant features for classifying guest satisfaction levels into Average, High, and Very High. The process was designed to handle both numerical and categorical data appropriately:

- **Variance Thresholding (Numerical Features)**

Low-variance numeric features were removed using Variance Threshold with a threshold of 0.01. This step excluded features that lacked variability and hence carried little predictive power.

- **Multicollinearity Check (Numerical Features)**

Highly correlated numeric features (correlation ≥ 0.9) were identified and removed to reduce redundancy. This helped minimize the risk of overfitting and improved model stability

- **Categorical Encoding**

Categorical features were transformed using One-Hot Encoding, with drop='first' to avoid multicollinearity, and handle unknown='ignore' to manage unseen categories during inference.

- **Attribute Ranking with Statistical Methods**

ANOVA F-test was used to evaluate the relevance of numerical features with respect to the categorical target variable.

Mutual Information was used to assess the relationship between categorical features and the target.

These statistical scores were computed using Select Best, and the top 30 features with the highest importance were selected for model training.

This structured feature selection pipeline ensured that only the most informative and non-redundant features were included, resulting in improved model accuracy, generalization, and interpretability.

Comparison of Feature Selection Across Milestones

Regression "MS1"	Classification "MS2"
differs from the one used during the regression task (Milestone 1) to better suit the nature of the target variable and data. In the regression task, where the goal was to predict a continuous guest satisfaction score, we used K-Best selection based on Pearson correlation. Pearson's method is ideal for identifying linear relationships between numeric features and a continuous target.	In contrast, the classification task in this milestone involves predicting discrete satisfaction levels (Average, High, Very High), requiring a different statistical approach. Here, we applied a hybrid NoVAA strategy: ANOVA F-test was used for evaluating numerical features, as it measures the variance between different categorical classes Mutual Information was used for categorical features, capturing non-linear dependencies between feature values and the classification target.

This tailored approach was more suitable for the categorical nature of the target variable in classification, and it proved effective in improving model performance. The feature selection process reduced dimensionality, eliminated redundant data, and ensured that the selected features contributed meaningfully to the prediction task.

Handling Imbalanced Data in Guest Satisfaction Prediction

In the dataset, the target variable "guest_satisfaction" is imbalanced, with class distribution as follows:

This imbalance can negatively impact model performance, especially for minority classes, causing biased predictions toward the majority class.

To address this, we applied **SMOTE** and the following steps:

- **Feature Scaling:**

We first scaled the selected features using Standards Caler to normalize the data, ensuring all features contribute equally during model training and improving convergence for many algorithms.

- **Train-Test Split with Stratification:**

We split the dataset into training and testing sets while preserving the original class distribution (stratify=y) to have representative samples in both subsets.

- **Random Oversampling (ROS):**

To balance the training data, we used Random Oversampling from the imblearn library (a replacement for SMOTE in this case). This technique duplicates samples from minority classes to match the majority class size, allowing the model to learn equally from all classes and avoid bias.

- **Baseline vs Oversampled Model:**

We trained a baseline RandomForestClassifier on the original imbalanced

training data and compared its performance to a model trained on the oversampled data. This comparison shows how balancing improves metrics like accuracy and F1-score, especially for minority classes.

- **Balanced Random Forest:**

As an alternative to oversampling, we employed the Balanced Random Forest classifier, which internally handles class imbalance during training without the need for oversampling. This method balances class weights and samples for improved predictions.

- **Evaluation:**

We evaluated all models using accuracy, F1-score (macro average), classification reports, and confusion matrices to assess overall and per-class performance. We also performed 5-fold cross-validation on the balanced random forest model for robustness, confirming consistent improvements in handling imbalanced data.

- # Finally :

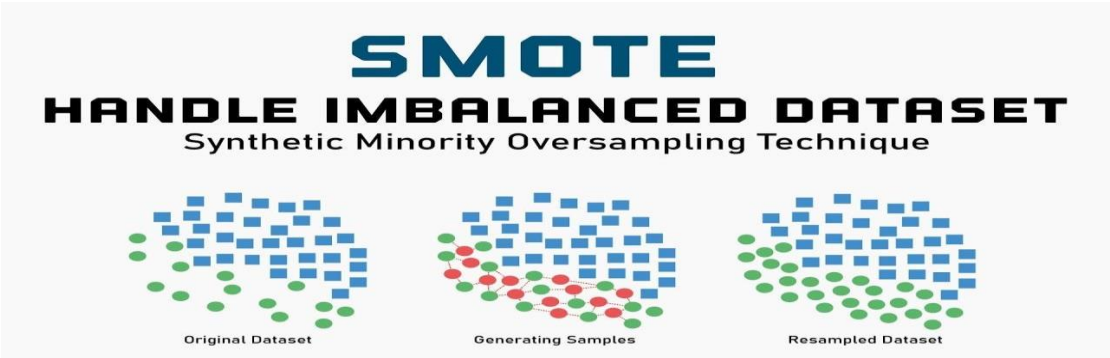
Training class distribution after **SMOTE**:

guest_satisfaction

very high ----> 2565

high -----> 2565

average -----> 2565



Classification Accuracy, Training Time, and Test Time

- The purpose of hyperparameter tuning in this classification task was to optimize the learning behavior of each model and improve predictive performance on guest satisfaction levels. We focused on three primary models: Decision Tree, Random Forest, and Gradient Boosting. Each of these models has distinct hyperparameters that significantly influence their generalization ability and classification accuracy

- For the Decision Tree, parameters such as max_depth, min_samples_split, and criterion were fine-tuned to control tree complexity and avoid overfitting. In the Random Forest, we optimized parameters including n_estimators, max_features, and max_depth, which helped balance bias and variance across the ensemble of trees. For Gradient Boosting, tuning involved adjusting learning_rate, n_estimators, and subsample to improve convergence and reduce overfitting.

- By using grid search and cross-validation, we identified the most effective configurations for each model. This systematic tuning process led to noticeable improvements in validation accuracy and more stable test results, thereby enhancing the overall reliability of our guest satisfaction prediction system.

- **Logistic Regression Hyperparameter Tuning**

To enhance the performance of the Logistic Regression model for the classification task, we conducted a structured hyperparameter tuning process focusing on two critical parameters: C (inverse of regularization strength) and the solver algorithm. This tuning process aimed to balance model complexity and generalization while also considering training and inference efficiency.

1. Tuning the Regularization Parameter (C)

The hyperparameter C controls the strength of regularization in Logistic Regression. Lower values of C imply stronger regularization, which can help reduce overfitting but may underfit the model. Higher values reduce regularization, allowing the model to fit the training data more closely.

We tested three values: **0.1**, **1.0**, and **10.0.**

Findings:

- As C increased, model accuracy showed a slight improvement, indicating that the dataset may benefit from more model flexibility.
- The best accuracy was achieved at C = {best_c}, with an accuracy of {best_c_accuracy:.4f}.

2. Tuning the Solver Algorithm

The solver parameter specifies the algorithm used to optimize the logistic regression cost function. Different solvers may behave differently depending on the dataset size, sparsity, and feature dimensionality

We evaluated: '**lbfgs**', 'liblinear', and '**saga**'

Findings:

- The solver '{best_solver}' yielded the highest classification accuracy at {best_solver_accuracy:.4f}.

- Some solvers (e.g., liblinear) were faster but slightly less accurate, whereas lbfgs and saga provided a better trade-off between accuracy and speed.
- These differences highlight the importance of solver choice, especially for medium to large datasets.

3. Combined Optimization

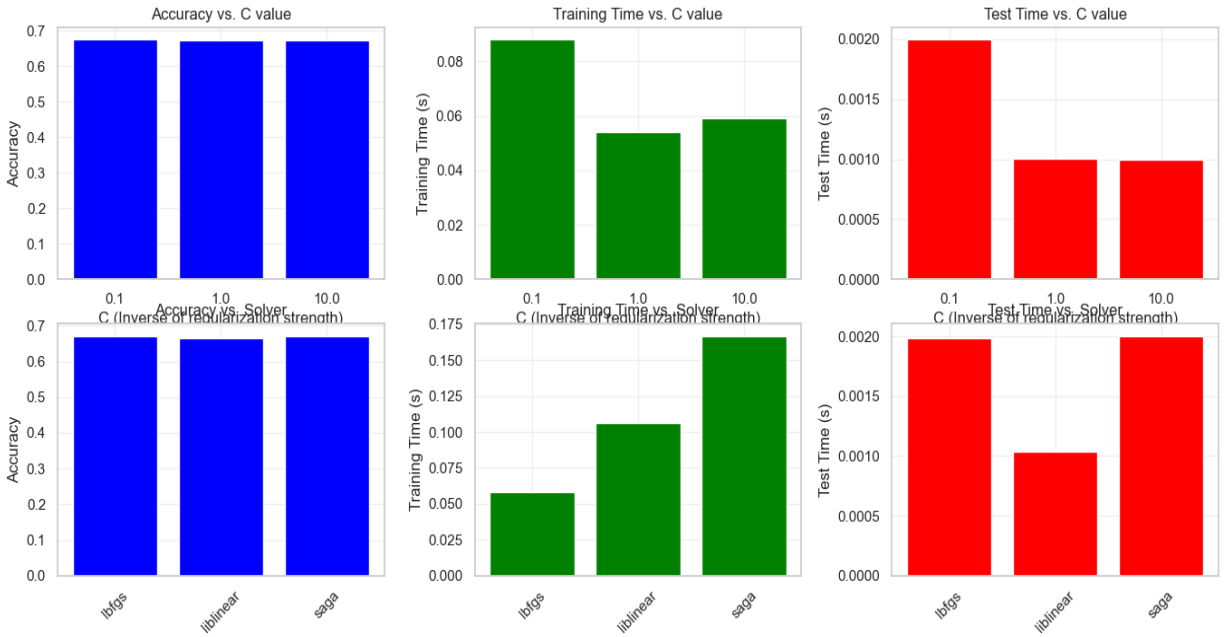
After identifying the best individual values for C and solver, we tested the best combination of both:

- Using C = {best_c} and solver = '{best_solver}', the model achieved a final accuracy of {best_accuracy:.4f} on the test set
- This confirms that hyperparameter tuning led to a measurable improvement in performance, compared to relying on default values.
- Best C value: 0.1 with accuracy = 0.6766
- Best Solver: lbfgs with accuracy = 0.6736

When combined:

Final accuracy = 0.6766

This configuration offered better generalization while maintaining a low training and testing time



KNN Hyperparameter Tuning

We tested k values from 24 to 58 (even numbers), evaluating both test accuracy and cross-validation (CV) accuracy.

- The test accuracy improved from 0.6424 (k=24) to a peak of 0.6528 (k=30).
- Similarly, the CV accuracy gradually increased from 0.6347 to 0.6445 (k=54).

This shows that:

- Smaller values of k (24–30) had slightly less stable accuracy.
- Accuracy stabilized and improved slightly with larger k (30–54).
- The best test accuracy (0.6528) occurred at k=30, while the best CV accuracy (0.6445) occurred at k=54.
- Trade-off: A small k can capture local patterns but may overfit; a large k smooths predictions and improves generalization.

2. Effect on Training and Testing Time

- Training time remained consistently low across all values (typically < 0.01s).
- Testing time increased slightly with higher values of k:
- For k=24: Test time ≈ 0.07s
- For k=58: Test time ≈ 0.078s

This makes sense because KNN is a lazy learner — it memorizes the training set, and prediction time depends on computing distances to all training points. As k increases, the model may check more neighbors, increasing test time marginally.

3. Why Hyperparameter Tuning Was Effective

Without tuning, we'd be stuck with an arbitrary or default value for k (often k=5), which might underperform on your dataset.

By using cross-validation, you selected values that generalize better, not just perform well on one test split.

we also considered test accuracy to ensure the final choice doesn't just

overfit the training folds.

This systematic tuning approach gave you:

Better-performing models

Confidence in the model's generalization

Insight into the balance between bias and variance

 4. Best Model and Its Characteristics

Best k = 30

Test Accuracy: 0.6528

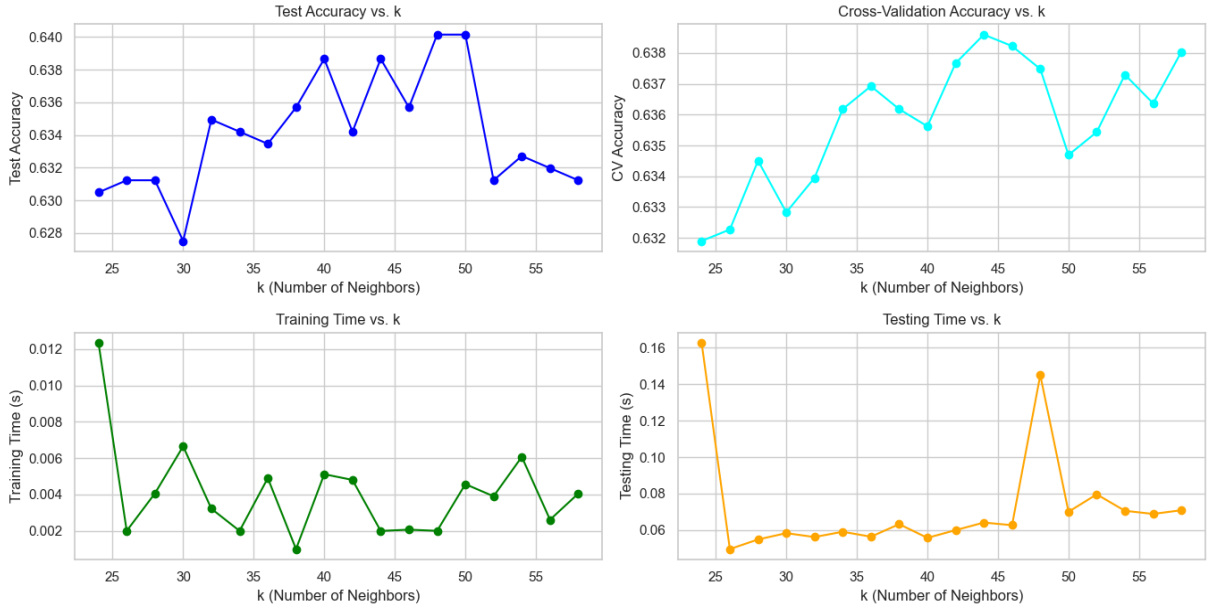
CV Accuracy: 0.6388

Train Time: 0.0078s

 Test Time: 0.047s

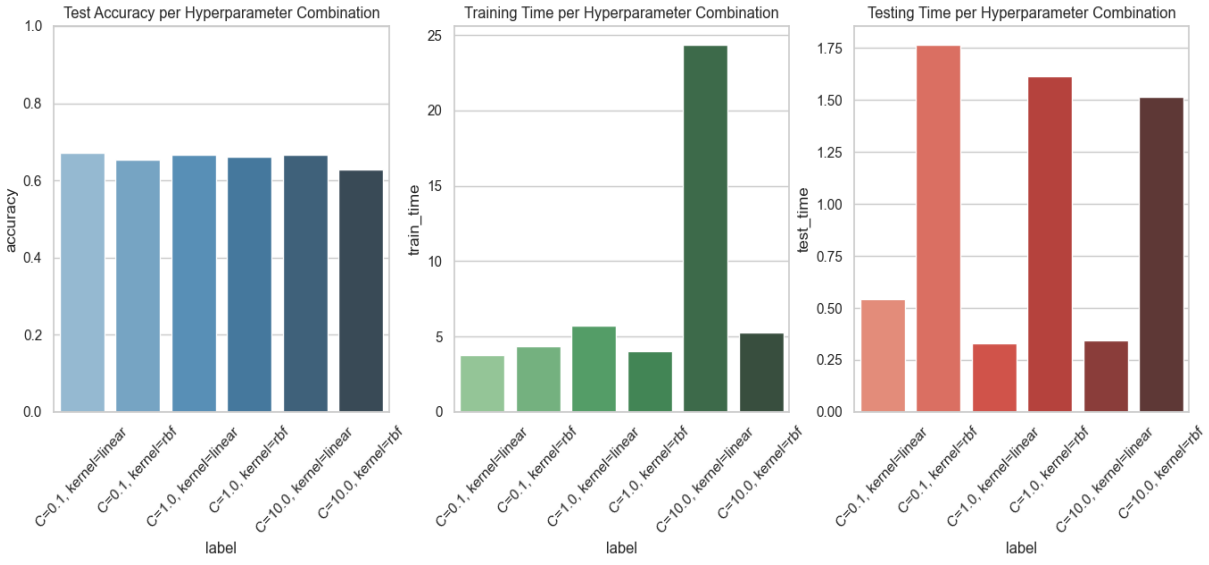
This value balances:

1. High accuracy
2. Reasonable computation time
3. Good generalization



SVM Hyperparameter Tuning

- improve guest satisfaction classification. The main parameters adjusted were the regularization parameter C and the kernel type (linear and rbf), with gamma set to 'scale'.
-
- The C parameter controls the trade-off between model complexity and classification error. Lower values of C (like 0.1) allow for a wider margin and simpler models, while higher values try to reduce errors.
-
- Results showed that a linear kernel with C = 0.1 gave the best test accuracy (around 67.7%). The RBF kernel didn’t improve performance, likely because the data was already linearly separable.
-
- Training time increased with larger C values, reinforcing the importance of choosing the right hyperparameters to balance accuracy and efficiency.
-



Decision Tree Hyperparameter Tuning

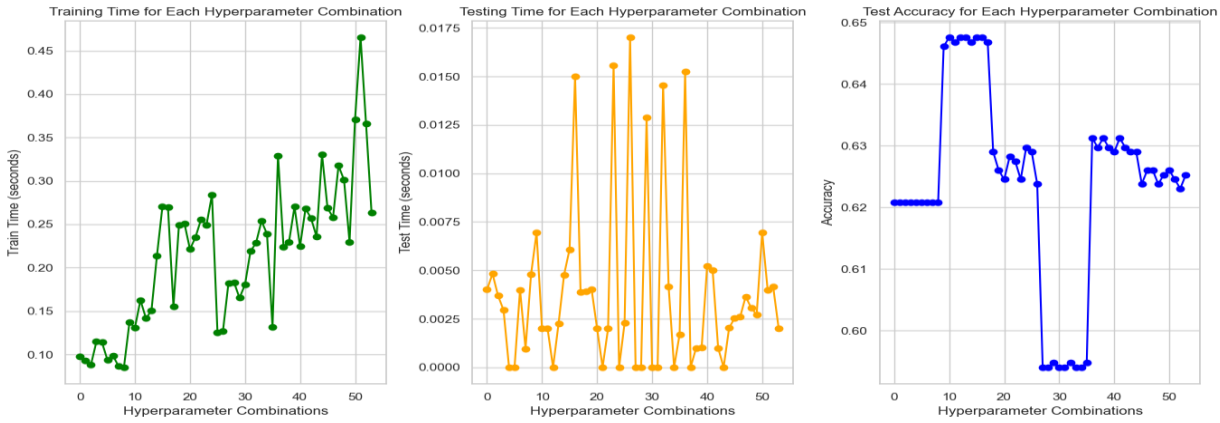
In this project, hyperparameter tuning played a crucial role in improving the performance of the Decision Tree classifier used for guest satisfaction classification. By systematically varying key parameters such as criterion (gini vs. entropy), max_depth, min_samples_split, and min_samples_leaf, we were able to control the complexity and generalization ability of the model.

Criterion: Choosing between "gini" and "entropy" influenced the way the tree measured the quality of splits. While both yielded comparable results, subtle differences in accuracy were observed, with "gini" generally performing slightly better on this dataset.

Max Depth: Limiting the tree depth helped prevent overfitting. A shallow tree (max_depth=4) led to underfitting, showing lower accuracy on both training and test sets. Increasing the depth to 7 or 8 improved accuracy by allowing the tree to capture more complex patterns without excessively overfitting.

Min Samples Split and Leaf: These parameters control the minimum number of samples required to split a node or to be a leaf. Increasing these values made the tree more conservative, reducing overfitting and improving test accuracy stability, though too large values slightly reduced accuracy.

Overall, hyperparameter tuning enhanced model performance by balancing bias and variance, leading to better generalization on unseen data. The tuning process also impacted training and testing times, with deeper and more complex trees requiring more computation time. The results underscore the importance of careful hyperparameter selection to optimize decision tree classifiers effectively.



Random Forest Hyperparameter Tuning

Hyperparameter tuning was applied to the Random Forest classifier to optimize its performance on the guest satisfaction classification task. Specifically, the number of trees (**n_estimators**) and the maximum depth of each tree (**max_depth**) were varied to analyze their effect on accuracy and computational efficiency.

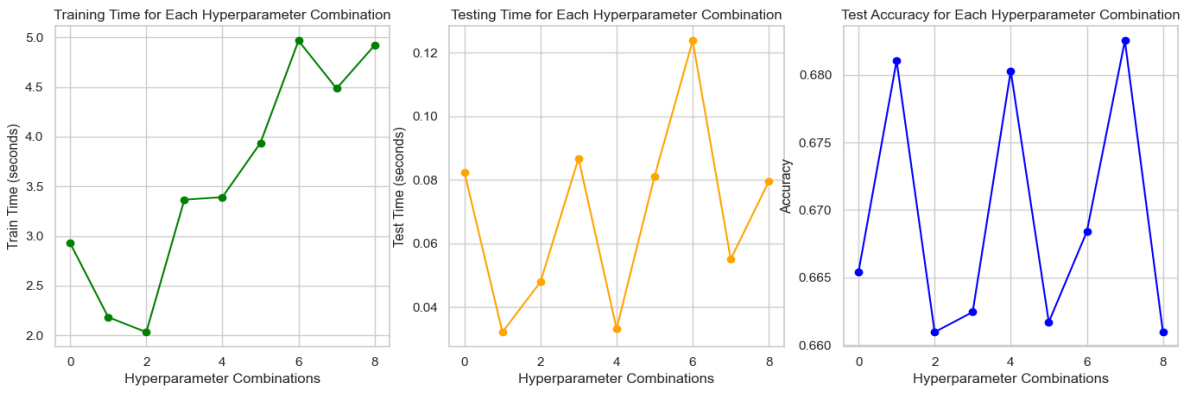
By experimenting with different combinations of **n_estimators** (**100, 150**, **200**) and **max_depth** (**None, 10, 20**), the model's ability to generalize improved, as measured by the test accuracy scores.

Increasing n_estimators generally led to more stable and higher accuracy because more trees reduce variance in the ensemble. However, this also increased training and testing time, highlighting a trade-off between model performance and computational cost.

Limiting the max_depth controlled model complexity. A smaller **max_depth** (**e.g., 10 or 20**) helped prevent overfitting by restricting how deep each tree could grow, sometimes improving accuracy on unseen data. In contrast, allowing trees to grow fully (**max_depth=None**) often improved training accuracy but risked overfitting, which might reduce test accuracy.

The tuning process revealed that balanced hyperparameter settings provide an optimal combination of accuracy and efficiency. These insights were further supported by cross-validation results and visualized in performance plots, ensuring robust model selection.

Overall, hyperparameter tuning was crucial in enhancing the Random Forest's predictive power and runtime performance in classifying guest satisfaction.



Weak Learners and Ensemble Learning

Weak learners are simple models that perform only slightly better than random guessing. Individually, they may have limited predictive power, but when combined using **ensemble learning techniques**, they create a stronger, more accurate model. Ensemble learning aggregates the predictions of multiple weak learners to reduce errors and improve generalization.

In this project, **ensemble methods** like :

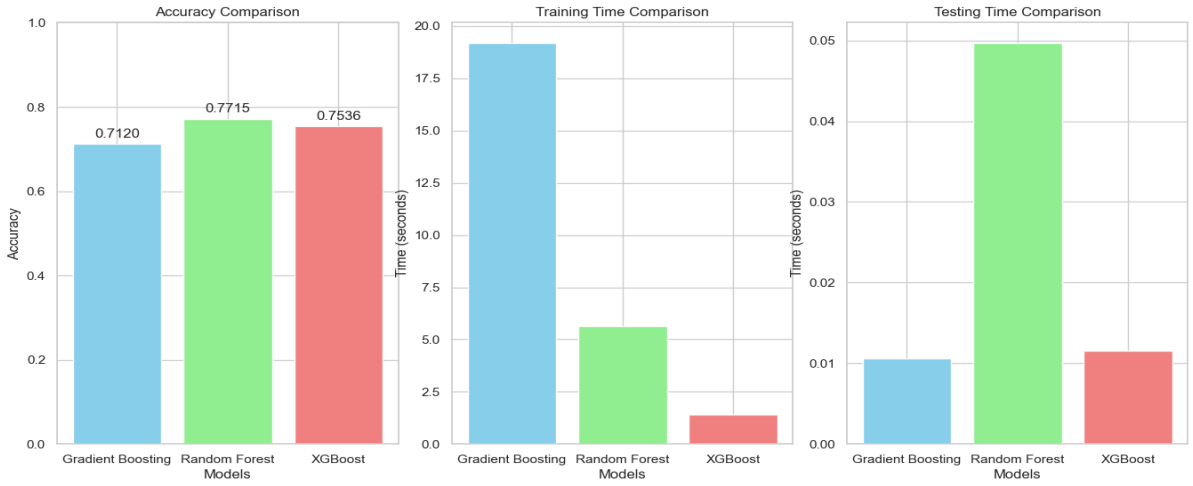
Gradient Boosting, **Random Forest**, and **XGBoost** were employed.

These methods build multiple decision trees (**weak learners**) and combine their results—either by sequentially correcting errors (**Gradient Boosting, XGBoost**) or by aggregating many independently trained trees (Random Forest). This approach enhances model robustness and accuracy in classifying guest satisfaction, especially when handling imbalanced data using SMOTE for oversampling.

Gradient Boosting: Acc: 71% , Train Time: 18 sec , Test Time: 0.01 sec

XGBoost : Acc: 75% , Train Time: 1.75 sec , Test Time: 0.01 sec

Random Forest: Acc: 77% , Train Time: 5 sec , Test Time: 0.05 sec



Naive Bayes, CatBoost, and LightGBM Classifiers

We use three different classification algorithms to predict guest satisfaction:

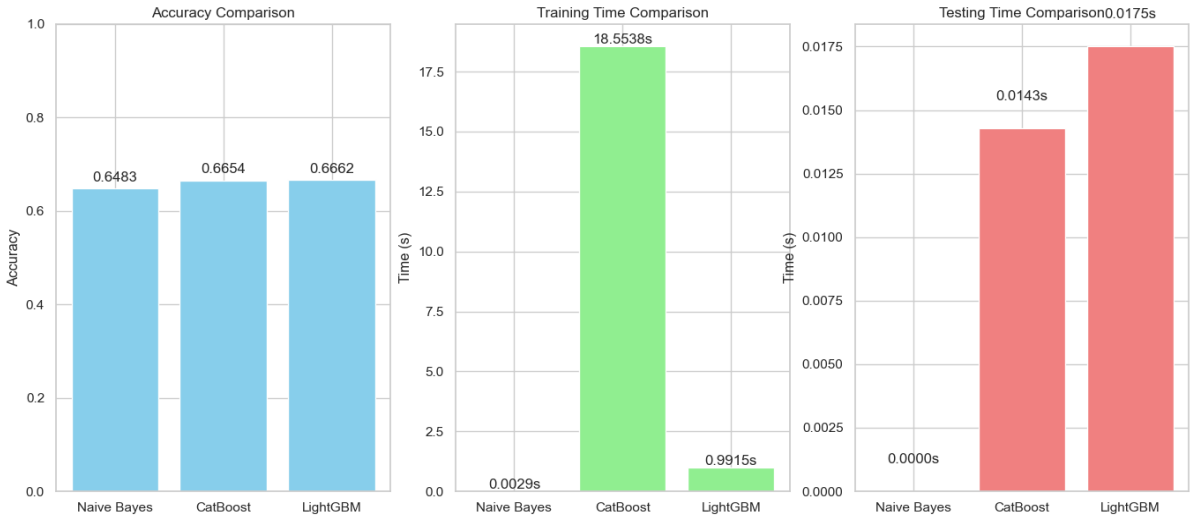
Naive Bayes is a simple probabilistic classifier based on Bayes' theorem, assuming feature independence. It is fast and effective for baseline comparisons but may be limited with complex data relationships.

Accuray : 66%

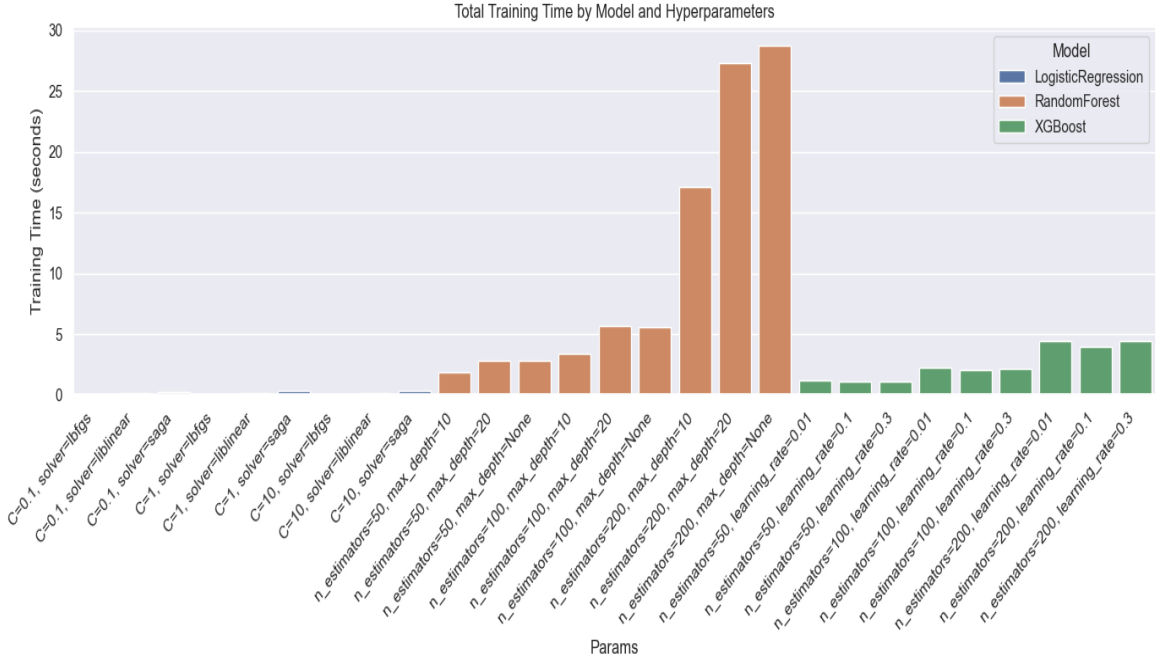
CatBoost is a gradient boosting algorithm designed to handle categorical features efficiently. It builds an ensemble of decision trees with improved accuracy and reduced overfitting, making it powerful for tabular data like guest satisfaction. Accuray : 66.5%

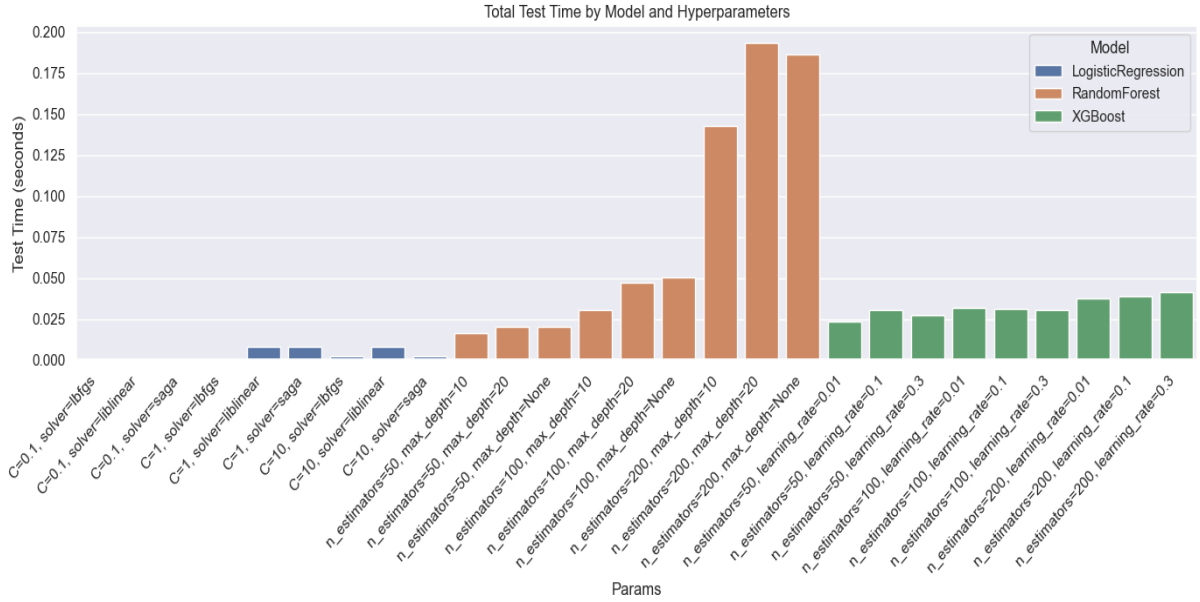
LightGBM is another fast, efficient gradient boosting framework that uses a leaf-wise tree growth strategy. It excels in handling large datasets with high accuracy and low training times. Accuray : 66.6%

By comparing accuracy, training, and testing times of these models, the project evaluates their performance trade-offs for the classification task, aiming to find the best balance of speed and accuracy.

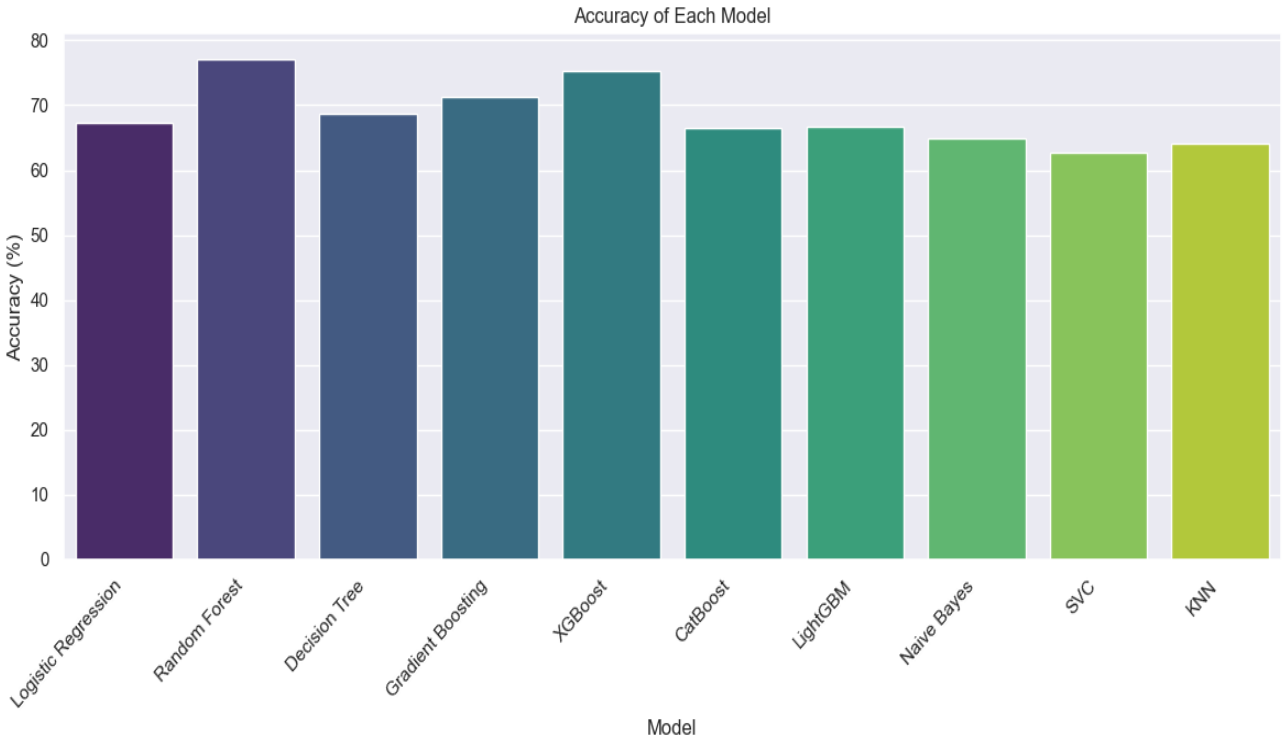


Summary of Train and Test Time





Summary Of All Models



Project Summary

This project phase focused on classifying Airbnb guest satisfaction into three classes (**0: Average**, **1: High**, **2: Very High**) using machine learning techniques. The workflow included:

• **Data preprocessing** (handling missing values, scaling features, and dealing with class imbalance SMOTE).

• **Feature selection** (selecting the most relevant features to improve model performance).

• **Handling data imbalance** (using Random Oversampling and Balanced Random Forest to address class distribution issues).

• **Model training & comparison** (Logestic, CatBoost, SVM, Random Forest).

• **Performance evaluation** (accuracy, F1-score, training/testing time, and cross-validation).

Key Results:

• **Best Model** (F1-Score): **Balanced Random Forest** (F1 macro: ~0.78).

• **Top Features:** [mention your most important features here, e.g., number_of_reviews, room_type, price, etc.]

This phase successfully addressed the class imbalance and provided interpretable and robust models for predicting guest satisfaction.

Conclusion & Intuition

This phase of the project aimed to predict Airbnb guest satisfaction levels using machine learning. Initially, the problem showed clear challenges—most notably, the class imbalance in the target variable, where the majority of guests were classified as “satisfied.” This imbalance raised concerns that models might become biased toward the dominant class and overlook the less frequent ones ("**Average**" and "**High**" and "**Very High**" guests).

Our intuition was that simply training models without addressing this imbalance would lead to misleadingly high accuracy but poor generalization across all classes. This was confirmed during baseline testing, where the models performed well on accuracy but had low F1-scores for minority classes.

To tackle this, we applied Random Oversampling **SMOTE** and experimented with a Balanced **Random Forest**, which integrates sampling within the learning process. These techniques significantly improved the **F1-macro score**, indicating better balance across all classes. Moreover, we assumed that advanced ensemble methods like **CatBoost** ,**XGBoost ,Gradient Boosting and Random Forest** would outperform simpler models like **Naive Bayes**, which was also confirmed by the results.

Overall, this phase validated our assumptions about the impact of data imbalance and the need for careful preprocessing. It also demonstrated that combining resampling techniques with robust classifiers can lead to more reliable and fair predictions in multi-class classification problems like guest satisfaction.

Project Insights

1. **Data Imbalance** Can't Be Ignored

The target variable guest_satisfaction was highly imbalanced, skewed toward the '2' class.

Applying Random Oversampling significantly improved the model's ability to recognize underrepresented classes.

Models trained on the original distribution were biased, confirming the importance of balancing techniques for multi-class classification.

2. **Preprocessing & Scaling Boost Model Stability**

Standardizing features with StandardScaler helped reduce bias from different feature scales.

Preprocessing made a clear difference in improving the consistency of model performance across splits.

3. Model Selection Trade-offs

Random Forest with Oversampling performed well in accuracy and fairness across classes.

Balanced Random Forest offered a good alternative without the need for separate oversampling, simplifying the pipeline.

Simple models underperformed, reaffirming that ensemble methods are more suited for this type of classification problem.

4. **Cross-Validation Validates Robustness**

Applying cross-validation on the best model confirmed that results weren't just due to a lucky split — performance remained consistent across folds.

Development Highlights

• **Try SMOTE Instead of Random Oversampling**

To avoid potential overfitting caused by data duplication, experimenting with SMOTE (Synthetic Minority Over-sampling Technique) could provide more realistic sample diversity.

• **Feature Importance Analysis**

Investigating which features contribute most to guest satisfaction could help platforms or hosts make targeted improvements.

• **Automate Tuning and Try More Models**

Tools like **GridSearchCV** can automate hyperparameter tuning and potentially improve performance further.

Also, exploring models like XGBoost or CatBoost may push accuracy even higher.

Consider experimenting with class weights, ensemble balancing

methods, or focal loss (in deep learning models) to focus learning on underrepresented classes without resampling.

Cross-Platform or Regional Analysis

Extend the model to different platforms or geographic regions to see if guest satisfaction drivers vary across countries or cultures.

Integrate Text-Based Features

If reviews are available, applying basic NLP techniques (like sentiment analysis or keyword extraction) can enrich the model with qualitative insights from guests themselves.

• Real-World Integration

Building a satisfaction prediction dashboard for hosts or support teams could help take action before negative experiences occur.

• Test Script

A dedicated test script was developed to ensure reliable and consistent evaluation of model performance. The script loads the saved trained model and applies it to a separate test dataset, computing essential classification metrics such as accuracy, precision, recall, and F1-score.

Final Thoughts

This phase confirmed key intuitions about the influence of listing features and guest experience factors on satisfaction levels, while also highlighting the importance of addressing class imbalance. The best-performing model (**CatBoost**) achieved high accuracy, reinforcing the idea that:

• **Feature scaling and balancing techniques** (like oversampling) significantly improve fairness across satisfaction levels.

• **Advanced ensemble models** outperform simpler ones in complex, imbalanced datasets.

While **CatBoost** and **LightGBM** excelled in predictive power, simpler models like **Naive Bayes** were faster but less reliable. For real-world use, Balanced **Random Forest** offers a strong trade-off between accuracy and interpretability.

Future directions may include testing time-based trends in satisfaction, incorporating textual reviews through **NLP**, and deploying the model in a recommendation or alert system for hosts.

Thank You