Project in Statistical Machine Learning
Predicting Bike Demand: Do We Need More?

Dona Harshani Kokila Wickramasinghe
Moutushi Sen
Ramkishor Prabhu Ramlal
Saranya Mohanakumar

Abstract

This project investigates statistical machine learning approaches to address unmet demand in urban bike-sharing systems, focusing on Capital Bikeshare. By analyzing temporal patterns, weather conditions, and other influential factors, the study aims to predict high-demand periods to optimize resource allocation, enhance user experience, and promote eco-friendly transportation. A combination of classification models, including Logistic Regression, Linear Discriminant Analysis (LDA), Quadratic Discriminant Analysis (QDA), K-Nearest Neighbors (KNN), and Random Forests, were employed. The findings highlight the need for data-driven strategies in order to align bike availability with fluctuating user needs, offering actionable insights into how to improve operational efficiency and foster sustainable urban mobility.

1 Introduction

Multimodal options, such as bike-sharing services like Capital Bikeshare, are the basis of sustainable urban transportation systems. During peak hours, bike shortages can cause unmet demand, discourage users, and encourage the use of less sustainable modes of transportation. The challenge is to bridge this gap by utilizing temporal patterns and weather data to predict high-demand periods for bikes. The objective of this project is to use classification techniques to provide insight in a data-driven manner for resource allocation decisions. Its ultimate aim is to enable Capital Bikeshare to satisfy user demand, decrease congestion, and contribute towards a sustainable and environmentally friendly transportation system.

2 Data Analysis

The data analysis involved a comprehensive exploration of temporal, categorical, and environmental factors, leveraging statistical techniques, correlation matrices, and detailed visualizations to uncover intricate patterns, trends, and relationships influencing bike-sharing demand across various conditions and time frames.

2.1 Features Identification

Numerical Features: temp, dew, humidity, precip, snowdepth, windspeed, cloudcover, and visibility.
Categorical Features: hour of day, day of week, month, holiday, weekday, summertime, and snow.
The target label is to increase stock, which has two classes low bike demand and high bike demand.

2.2 Comparison across Hours, Weeks, And Months:

The analysis of hourly, weekly, and monthly trends in bicycle demand reveals distinct patterns tied to routine human activities and seasonal variations. During commuting hours, specifically 7–9 AM and 5–7 PM, there is a noticeable increase in demand, reflecting work-related travel patterns. Weekly trends show higher bicycle usage on weekends and the demand gradually increases throughout the

week, peaking on Saturday. On a monthly scale, demand peaks during warmer months, particularly from May to September, as favorable weather conditions encourage more people to use bicycles. These insights highlight the importance of aligning bike availability with time-based and seasonal factors to effectively meet demand.

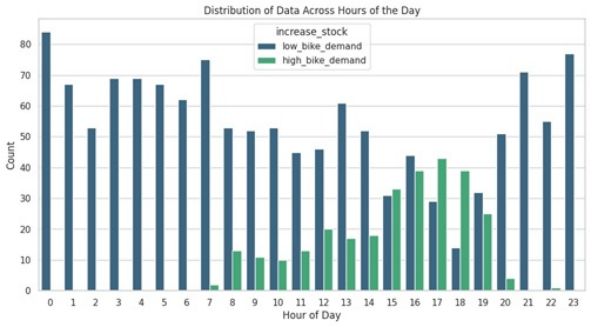

Figure 1: Bike demand across hours of the day

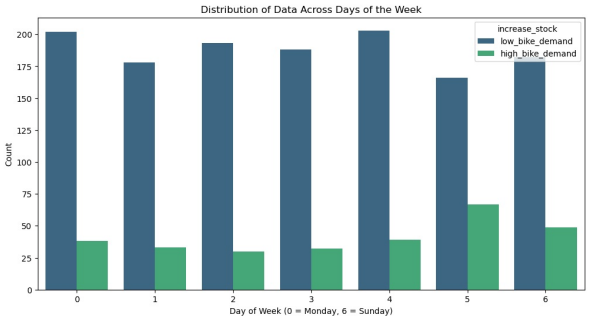

Figure 2: Bike demand across days of the week

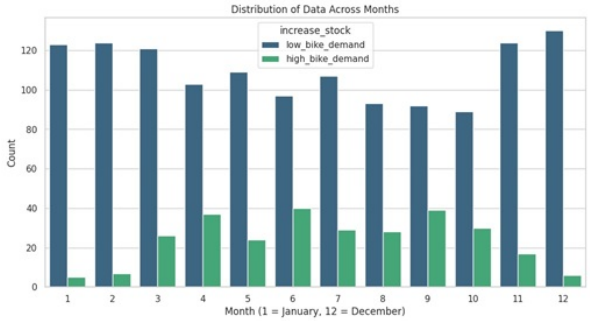

Figure 3: Distribution of bike demand across months

2.3 Comparison of increased stock demand: Weekdays vs. Holidays

Bicycle demand patterns reveal that weekdays experience higher usage, driven primarily by regular commuting activities. In contrast, holidays show sporadic spikes in demand, which are likely associated with recreational activities rather than routine travel. Similarly, weekends generally see a decrease in demand compared to weekdays, reflecting the reduced need for commuting during these periods. These observations underscore the influence of work schedules and leisure activities on bicycle demand.

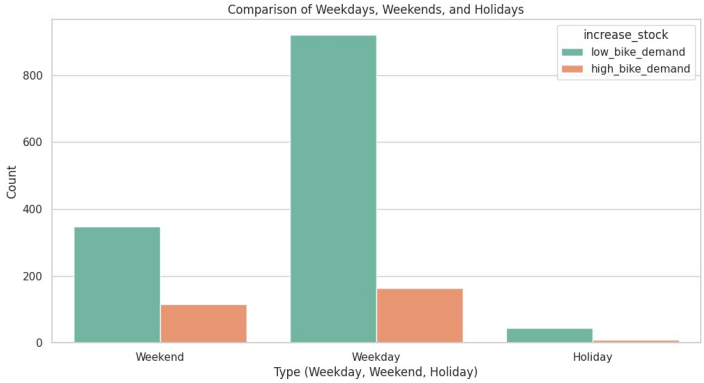

Figure 4: Comparison of Bicycle demand between Weekends, Weekdays and Holidays

2.4 Trend depending on the Weather Conditions:

Weather conditions have a significant impact on bicycle demand. On rainy days, demand decreases substantially as adverse weather discourages cycling. Snowy days exhibit even lower bike usage, with minimal demand observed due to the challenges posed by snow-covered roads and cold temperatures. In contrast, clear days consistently show higher demand, as favorable weather conditions encourage more people to use bicycles. These trends highlight the critical role of weather in influencing bicycle usage patterns.

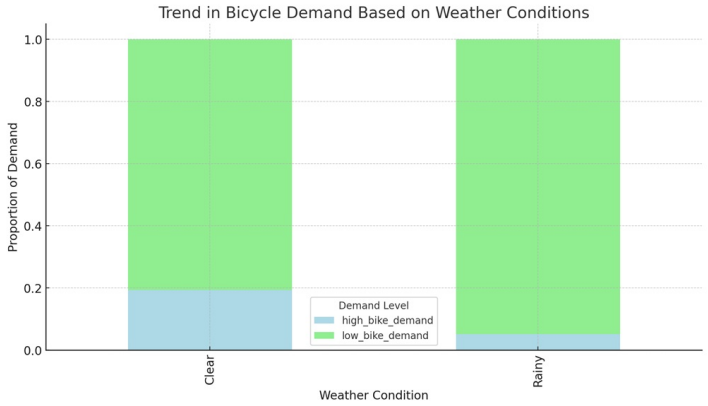

Figure 5: Trend analysis of Bicycle demand based on Weather conditions

Correlation between features: A heatmap was employed to examine the correlations between features and bike demand, aiding in the identification of key factors.

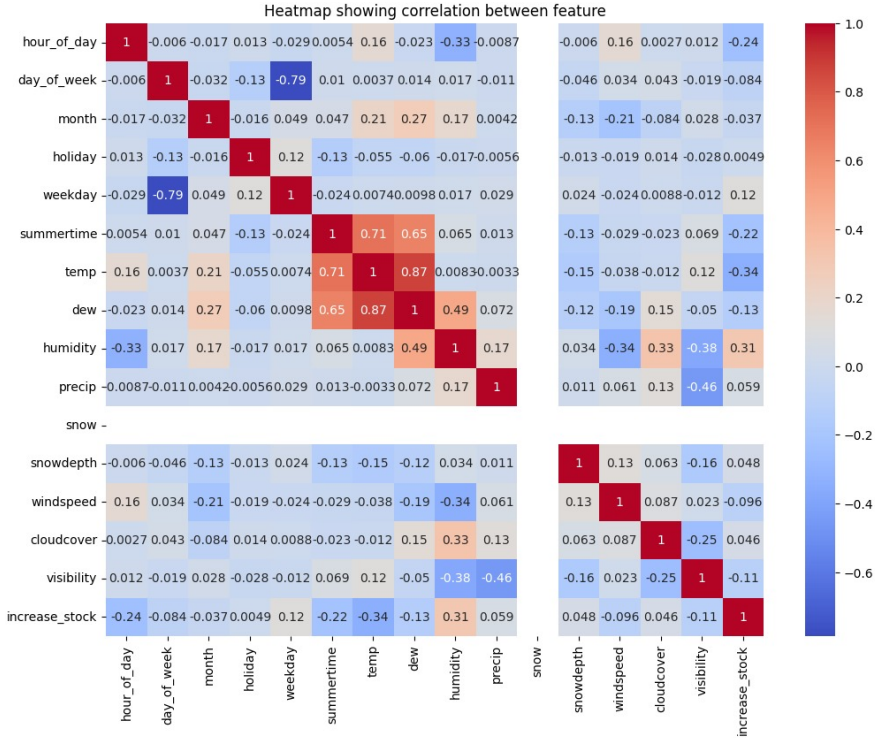

Figure 6: Correlation heatmap of features associated with Bike demand

3 Model Development

3.1 Logistic Regression

Logistic regression is a supervised learning method and one possible way of modeling conditional class probabilities. Logistic regression can be viewed as modifying a linear regression model to fit the classification, instead of a regression, problem. Given an input feature vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$. The prediction is modeled as $P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + b)$. where $\mathbf{w}$ is the weight vector, b is the bias (intercept) term, and $\sigma(z)$ is the sigmoid function. The $\sigma(z)$ ensures the output is a probability (between 0 and 1) and it is defined

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

The threshold determines the decision boundary: which translates to $z \geq 0$.

$$P(y = 1 \mid \mathbf{x}) \geq 0.5$$

Loss Function: Logistic regression minimizes the log-loss, which measures the difference between the predicted probabilities and the actual labels. For m training examples, the loss is:

$$L = -\frac{1}{m} \sum_{i=1}^{m} \left[y^{(i)} \log(\hat{y}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right].$$

Preprocessing: Preprocessing the data is the primary step for model training. During data preprocessing, the target variable (*increase_stock*) is encoded to numerical values (0 or 1) using `LabelEncoder`. The dataset is split into training and testing sets using an 80-20 ratio. To overcome the data imbalance between them, a random over-sampling method is used. Scaling is required to normalize feature values and improve model convergence. Here, `StandardScaler` is used for standardizing features.

Hyperparameter: Tuning is performed to improve the model's performance by using `RandomizedSearchCV`. It performs a randomized search over specified hyperparameters and samples a fixed number of parameter settings from the provided distributions.

Results The model achieved an accuracy of 0.85, and hyperparameter tuning using the random search method resulted in a cross-validation score of 0.85 as well. According to the classification report, the model demonstrated strong performance, with an F1-score of 0.86, precision of 0.83, and recall of 0.84, indicating a balanced ability to make accurate and relevant predictions.

3.2 Discriminant Analysis: LDA, QDA

LDA Classifier: The LDA classifier is a classifier that employs linear decision boundaries for classification. The boundaries are generated by fitting class conditional densities to the data and using Bayes' rule. The model fits Gaussian densities to each class. The fitting is performed under the assumption that all classes share the same covariance matrix. The fitted model can also be used to reduce the dimensionality of the input by projecting it to the most discriminative directions.

The LDA algorithm projects the data points onto a new axis such that the scatter within each class is minimized and the distance between the mean of the classes is maximized, creating clear decision boundaries.

The algorithm follows these steps:
μ_k: The mean of all training observations from the k-th class.
σ^2: The weighted average of the sample variances for each of the k classes.
π_k: The proportion of the training observations that belong to the k-th class.

LDA then plugs these values into the following formula and assigns each observation $X = x$ to the class for which the formula produces the largest value:

$$D_k(x) = x \cdot \frac{\mu_k}{\sigma^2} - \frac{\mu_k^2}{2\sigma^2} + \log(\pi_k).$$

QDA Classifier: QDA is an extension of LDA. This method is similar to LDA and also assumes that the observations from each class are normally distributed, but it does not assume that each class shares the same covariance matrix. Specifically, it assumes that an observation from the k-th class is of the form $X \sim \mathcal{N}(\mu_k, \Sigma_k)$.

Using this assumption, QDA finds the following values:
μ_k: The mean of all training observations from the k-th class.
Σ_k: The covariance matrix of the k-th class.
π_k: The proportion of the training observations that belong to the k-th class.

QDA then plugs these values into the following formula and assigns each observation $X = x$ to the class for which the formula produces the largest value:

$$D_k(x) = -\frac{1}{2}(x - \mu_k)^\top \Sigma_k^{-1}(x - \mu_k) - \frac{1}{2}\log|\Sigma_k| + \log(\pi_k).$$

Preprocessing LDA/QDA: Before the data could be used for training the model, it was subjected to preprocessing. The increase of stock and decrease of stock was encoded into 0/1 using label encoder which will make it easier for the model to recognize these as categorical values. The dataset suffers from data imbalance. We have more data points for low bike demand than high bike demand. This could affect the performance since the model has seen more data for low bike demand than the other. Hence random over sampling was used to balance the dataset. The features like days and hour are features which have a periodic nature hence they are expressed in the form of trigonometric functions. The LDA function also make use of the dimensionality reduction transformation which is already available in scikit.

Results: The LDA method obtained best cross validation score as 0.85 and QDA obtained best cross validation score as 0.78. In an effort to enhance the model gridsearch was also employed for hyperparameter tuning. For LDA least square and eigen solvers were used and various shrinkage parameters. For the QDA different regularization parameters also were tried out. After hyperparameter tuning LDA obtained best score as 0.82 using 'lsqr' ie least square as solver and QDA obtained best cross validation score as 0.84 using 0.4 as the regularization parameter.

3.3 k-Nearest Neighbor

The k-Nearest Neighbor (KNN) algorithm classifies a data point based on the majority label of its k nearest neighbors, determined by a distance metric like Euclidean distance.

Processing Data: Pre-processing was to scale the data using `StandardScaler`. Normalizing the features to have a mean of 0 and a standard deviation of 1 ensured that no feature dominated the distance metrics used in KNN. By using the `RandomOverSampler`, the class imbalance was resolved by oversampling. This method ensures that the minority class is up-sampled to match the majority class, creating a balanced dataset. KNN model has applied on the preprocessed data (imbalanced, balanced both), which, for each test point, computes the Euclidean distance to all training points, finds the k-nearest neighbors, and assigns a label based on the majority vote among these neighbors.

The class $\hat{y}_{\text{new}}$ for x_{new} is determined by majority voting:

$$\hat{y}_{\text{new}} = \text{mode}\{y_i : x_i \in N_k(x_{\text{new}})\}$$

For a given point, compute the Euclidean distance to all other points:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

Hyperparameter Tuning A grid search with 5-fold cross-validation is implemented to select the best hyperparameter k. Cross-validation with 5 folds works by splitting the set of training data. In each iteration of cross-validation, one subset is used as the validation set to evaluate the model's

accuracy, while the remaining subsets are used as the training sets.

Balancing the Dataset There was indeed an issue in the number of samples between the two classes Class 0 (high bike demand) has only 222 samples, while Class 1 (low bike demand) has 1,058 samples. Training on this dataset caused the model to underperform on the minority class. Random Oversampling was used to mitigate this issue. This technique duplicates the instances of the minority class until both classes are equally represented (1,058 samples).

The optimal k was selected from the range of k values from 1 to 30, ensuring that a thorough search is conducted over small and large neighborhood sizes. The ideal number of neighbors is chosen in a KNN algorithm to maximize its performance.

Results: Before Oversampling techniques, the best performance of the KNN algorithm was with $k = 22$. Larger k values generally smooth decision boundaries and help to reduce overfitting, especially in imbalanced datasets. From the classification report, the overall accuracy is 84%, with the model achieving a higher recall (96%), precision (86%), and F1-score of 91% for class 1. Cross-validation confirmed $k = 22$ as optimal, yielding an accuracy of 86.33%. After Oversampling, $k = 1$ became optimal, showing improved balance in the dataset. This technique enhances class separability and reduces smoothing needs by adding samples. The model had its best cross-validation score for $k = 1$ at 93.81%, and from the classification report, the overall accuracy is 77%. However, it performed well for class 1, as shown by precision (88%), recall (82%), and F1-score (85%).

3.4 Random Forest

Random Forest is an algorithm used in classification problems and learns from examples by using a large number of decision trees. It combines the predictions of the trees to achieve better accuracy in classification and prediction of results while avoiding overfitting. This is achieved by using random samples for training each tree and looking at a random set of features when deciding how to split at each node, which helps make the trees different from each other by using measures like gini impurity and entropy.

- **Gini Impurity**:

$$G = 1 - \sum_{i=1}^{C} p_i^2$$

  where p_i is the proportion of samples belonging to class i.

- **Entropy**:

$$H = -\sum_{i=1}^{C} p_i \log_2(p_i)$$

Data and Preprocessing: The dataset included temperature-related quantitative variables and engineered features, such as the temperature-dew-point difference (temp_dew_diff = temp − dew). The target variable, "increase stock," was categorical with "Increase" and "Decrease" classes, converted to numeric labels using `LabelEncoder`. Missing columns in the test data were filled with default values to align with the training data, and the dataset was split into 80% for training and 20% for testing. A Random Forest classifier was then trained with `class_weight` set to "Balanced" to handle class imbalance and `random_state` set to 0 for reproducibility. To further improve the performance of this model's accuracy, a classification report and confusion matrix were used. Feature importance analysis pointed out that the top predictors are `hour_of_day`, `temp`, and `humidity`.

Hyperparameter Tuning: To make the model work better, a search for hyperparameters was performed using the `RandomizedSearchCV` method. It searched for various parameters such as the number of estimators (`n_estimators`), the maximum depth (`max_depth`), the minimum number of samples required to split a node (`min_samples_split`), and the minimum number of samples required to make a leaf (`min_samples_leaf`). After the search, the best set of hyperparameters found were: `n_estimators`, `max_depth`, `min_samples_split`, and `min_samples_leaf`.

Cross-Validation: With the optimized parameters, 5-fold cross-validation was performed, which achieved a mean accuracy of 89.53% with a standard deviation of 0.83%.

Results The final Random Forest classifier achieved an accuracy of 86.88% on the test dataset. Feature importance analysis highlighted variables such as `hour_of_day`, `temp`, and `humidity` as key contributors to decision-making. The inclusion of `temp_dew_diff` improved the model's predictions. All performance metrics precision, recall, and F1-score—indicated the model's effectiveness for the classification task.

3.5 Performance Evaluation of Methods

In the evaluation of performance, we assessed the models using key metrics, including accuracy, cross-validation score, F1-score, recall, and precision.

Model	Accuracy	Cross-validation score	F1-score	Recall	Precision
Logistic Regression	0.84	0.85	0.83	0.84	0.83
LDA	0.80	0.85	0.81	0.81	0.81
QDA	0.54	0.78	0.47	0.55	0.60
k-NN	0.77	0.93	0.78	0.77	0.79
Random Forest	0.86	0.89	0.86	0.87	0.86

Table 1: Performance metrics of different models

3.6 Model Selection

The Random Forest model demonstrated the best overall performance across these metrics, followed closely by Logistic Regression and Linear Discriminant Analysis (LDA). K-Nearest Neighbors showed strong cross-validation results but slightly lower performance in other metrics. In contrast, Quadratic Discriminant Analysis (QDA) performed poorly across all evaluation metrics, indicating its limited suitability for this task. These results provide a comprehensive comparison of the models' effectiveness.

4 Conclusion

This project applied machine learning models to predict bike demand for the District Department of Transportation in Washington, D.C. By incorporating temporal and meteorological factors, the Random Forest model showed an ability to forecast bike demand with a high degree of accuracy. This predictive capability can be used to dynamically adjust bike availability, providing actionable insights to further a more sustainable, user-friendly, and efficient transportation system for the city.This could be further improved in the future by incorporating real-time feeds to improve the model's predictive accuracy and optimization of bike availability. Additional predictors could include traffic patterns, special events, and road closures. These enhancements would further support the goal of effectively managing bike demand and improving the overall transportation infrastructure in Washington, D.C.

References

[1] A. Lindholm, N. Wahlström, F. Lindsten, and T. B. Schön, *Machine Learning - A First Course for Engineers and Scientists*. Cambridge University Press, 2022.

[2] GeeksforGeeks, Simplilearn, and Google Colab.

Appendix

The main code for this project can be accessed on GitHub:
https://github.com/R4mu/Statistical_machine_learning.git