Data-Driven Demand Forecasting for Smarter Bike-Sharing Management

Muhammad Iman Khalifa (301417007)

Introduction

In recent years, bike-sharing systems have emerged as a popular and sustainable mode of transportation in urban areas. Apart from their flexibility, they offer an eco-friendly alternative to traditional commuting, help alleviate traffic congestion, and promote healthier lifestyles through active mobility. However, the success of these systems depends heavily on being able to accurately forecast demand. If too many bikes are sent to low-demand areas or not enough are available during peak times, it leads to inefficiencies, lost revenue, and frustrated users.

For a bike-sharing company, the stakes are high: supply–demand imbalances can cause frequent shortages in high-demand areas and surpluses in low-demand zones, frustrating customers and reducing ridership. At the same time, operational inefficiencies and cost overruns can occur when bikes are redistributed unnecessarily, driving up labor, fuel, and logistics expenses. Accurate demand forecasting tackles both challenges by enabling smarter pricing strategies, efficient bike allocation, targeted marketing during peak periods, and data-driven investment decisions for fleet growth or maintenance. It can also help identify emerging high-demand zones, allowing the company to strategically position docking stations and secure market share before competitors do.

Beyond operational efficiency, accurate demand forecasting can also prevent revenue loss and missed opportunities by ensuring bikes are available when and where customers want them most — such as during commuter rush hours, tourism surges, and special events. It further supports asset management and long-term profitability by minimizing unnecessary relocations that cause wear-and-tear, thereby reducing maintenance costs and extending bike lifespan. Predictive insights can also support partnerships with local businesses and municipalities, enabling co-branded docking stations or event-based promotions that boost both ridership and brand visibility. All of these benefits strengthen margins, improve market competitiveness, and contribute to a more sustainable business model.

This project focuses on predicting the number of bike rentals per hour in London using weather and time-related variables. The ability to make accurate predictions has practical economic value — it allows operators to manage supply more efficiently, reduce operational costs, and improve the customer experience. For city planners, it also means better-informed decisions about infrastructure and resource allocation. The goal of this paper is to build a model that helps answer: Can we predict hourly bike demand in London based on weather conditions and time features?

Data

The dataset contains hourly records of bike-sharing activity in London, including weather and time-related information.This dataset has 17,414 hourly observations from January 2015 to January 2017. It contains no missing values. The `timestamp` variable is further decomposed into derived features such as `hour`, `weekday`, and `month` for analysis and modeling.

The dataset was randomly split into an 80% training set and a 20% holdout test set to enable unbiased performance evaluation. The holdout set was set aside prior to model training and was not used in any part of the model selection, parameter tuning, or cross-validation process.

Model Selection

This project compares the predictive performance of three models taught in class: `lm`, `randomForest`, and `xgboost`. These models were selected to represent a range of model complexity, from linear to highly flexible machine learning methods.

Model	Description
`lm`	Linear regression serves as a simple, interpretable baseline model. It assumes a linear relationship between predictors and the response.
`randomForest`	An ensemble of decision trees using bagging. It reduces variance and handles nonlinear interactions well, especially with categorical predictors.
`xgboost`	A boosting algorithm that builds trees sequentially to minimize residual error. Known for high performance on structured data.

Each model was evaluated using **Mean Squared Error (MSE)** on a **held-out test set** (20% of the data) to measure prediction accuracy. Additionally, **training vs. test MSE** was compared to assess overfitting or underfitting.

Selection Criteria

- All models were trained on the same 80% training set using a fixed random seed.
- Hyperparameters for `randomForest` (e.g., `ntree`, `mtry`) and `xgboost` (e.g., `nrounds`, `eta`, `max_depth`) were tuned using cross-validation.
- The model with the **lowest test MSE** was selected as the final model.

This model comparison approach is consistent with the model evaluation framework described in *An Introduction to Statistical Learning* (James et al., 2021), which recommends starting with simple models and evaluating performance improvements as complexity increases.

Based on previous empirical evidence and expected performance: - `lm` provides a useful baseline - `randomForest` improves prediction by reducing variance - `xgboost` is expected to perform best due to its ability to model nonlinear interactions and correct residual errors

The final model will be selected based on test set performance and model interpretability.

Model Performance

Reported test results are based solely on the 20% holdout set, which was not involved in model training or tuning. This ensures that the performance metrics reflect the model's ability to generalize to unseen data.

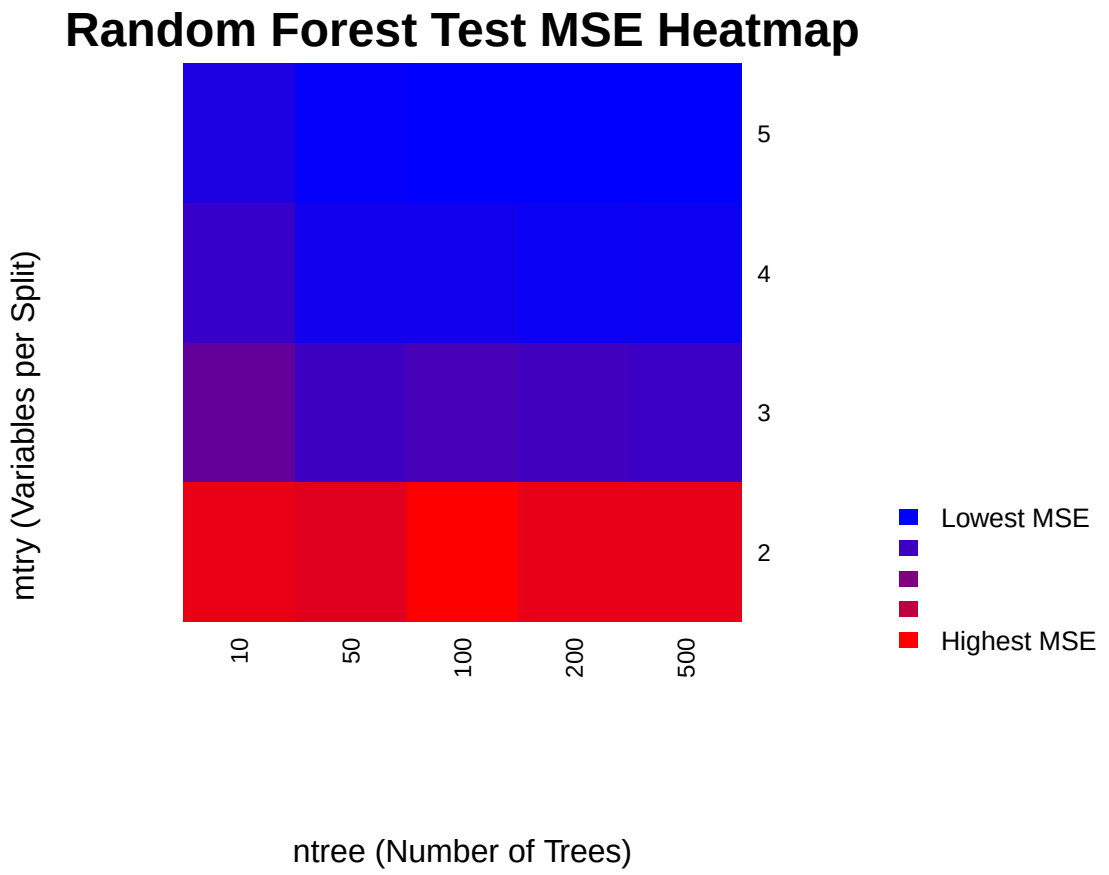
OLS

```
##     Set      MSE     RMSE
## 1 Train 815013.1 902.7807
## 2  Test 836479.4 914.5925
```

The Ordinary Least Squares (OLS) model served as a simple baseline, providing an interpretable starting point for evaluating more complex models. On the training set, the OLS model produced an RMSE of approximately 903 bikes per hour, and on the test set, about 915 bikes per hour. This relatively high error indicates that a purely linear approach struggles to capture the nonlinear and interaction effects present in the data. While OLS offers transparency in terms of coefficient interpretation, its limited flexibility makes it less suitable for operational decision-making in this context, where accurate hour-by-hour predictions are crucial for optimizing bike availability.

Random Forest

```
## Time difference of 6.335063 mins
```



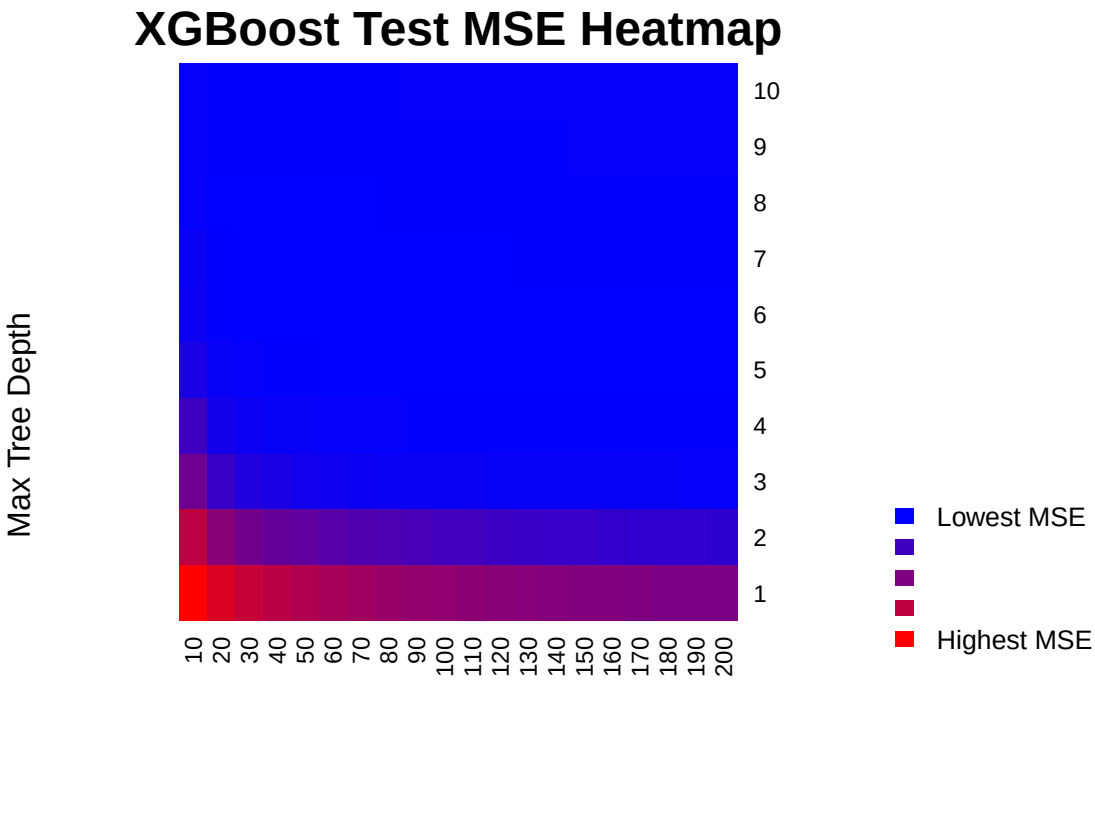

```
## Best Test MSE at mtry = 5 and ntree = 500
```

Unlike OLS, which does not require parameter tuning, the Random Forest model's accuracy depends on selecting the right settings for key parameters such as mtry (the number of variables considered at each split) and ntree (the number of decision trees grown). Tuning these parameters allows the model to balance the trade-off between capturing enough patterns in the data and avoiding excessive complexity that could harm predictions on new data.

The heatmap provided a visual guide to identify the parameter combinations that minimized prediction error. Darker zones in the plot indicated better-performing settings, helping us zero in on mtry = 5 and ntree = 500 as the best balance. This visual approach makes it easier for decision-makers to see how model performance changes across the parameter space without needing to understand the underlying algorithms.

```
##     Set      MSE     RMSE
## 1 Train 15671.47 125.1857
## 2  Test 64777.32 254.5139
```

With the tuned parameters applied, the Random Forest model achieved an RMSE of about 125 bikes/hour on the training set and 255 bikes/hour on the test set. This marks a substantial improvement over OLS, indicating the model's ability to capture complex relationships in the data. However, the noticeable gap between train and test results suggests mild overfitting, meaning the model fits the training data more closely than the unseen test data. Despite this, the accuracy gain is significant enough to inform more efficient bike allocation strategies, reducing idle bikes in low-demand areas and avoiding shortages in high-demand periods.



XGBoost

```
## Lowest Test MSE at max_depth = 6 and nrounds = 110
```

XGBoost requires tuning parameters such as max_depth (how deep each decision tree can grow) and nrounds (the number of boosting iterations). This process is more intensive than OLS and even Random Forest, as boosting builds trees sequentially, with each one improving on the errors of the last. Tuning is essential because the wrong settings could either underfit (too simple) or overfit (too complex) the data.

The heatmap for XGBoost illustrated how prediction error varied across different combinations of max_depth and nrounds. This allowed us to quickly spot the optimal values (max_depth = 6, nrounds = 110), where the model produced the lowest test error. The visual also highlighted diminishing returns — after a certain point, adding more depth or rounds provided little to no benefit, guiding us toward an efficient model configuration.

```
##     Set      MSE     RMSE
## 1 Train 30332.85 174.1633
## 2  Test 57371.31 239.5231
```

With optimal tuning, XGBoost delivered the best results among all models tested, achieving an RMSE of approximately 174 bikes/hour on the training set and 240 bikes/hour on the test set. Not only is this more accurate than both OLS and Random Forest, but the smaller train–test gap suggests stronger generalization to unseen data. This means the model is less likely to break down when applied to new scenarios, making it highly reliable for anticipating demand surges caused by weather changes, rush hours, or special events. The improved precision can directly translate into better fleet deployment, reduced operational costs, and increased customer satisfaction.

CEO Explanation

From a business standpoint, these results have direct economic implications. The progression from OLS to Random Forest to XGBoost illustrates how advanced modeling can cut forecast errors by hundreds of rentals per hour. For a bike-sharing operator, this translates into fewer missed rental opportunities in high-demand areas, less idle inventory in low-demand zones, and reduced costs from unnecessary bike movements.

Implementing the XGBoost model in operational systems would allow the company to anticipate demand with greater precision, supporting targeted bike deployment, optimized maintenance scheduling, and dynamic pricing during high-demand periods. Over time, these improvements could lead to higher ridership, stronger customer retention, and improved profitability, while also informing strategic decisions such as where to expand docking infrastructure or form local business partnerships. In short, the analysis provides the CEO with a data-backed path to both operational efficiency and long-term competitive advantage.

Recommendations for the CEO

To translate these findings into action and directly address the company's key challenges, the following measures are advised:

- Shortages in high-demand areas → Integrate the model's hourly forecasts into an automated rebalancing system that prioritizes bike transfers to stations predicted to spike, particularly during rush hours and large events.
- Surpluses in low-demand zones → Use demand forecasts to cap initial deployments and shift excess bikes to higher-demand areas in near real time.
- Operational inefficiencies → Adopt predictive maintenance scheduling based on actual usage patterns instead of fixed time intervals, reducing labor, fuel, and logistics costs.
- Revenue loss and missed opportunities → Implement dynamic pricing during forecasted peaks, run targeted marketing in slow periods, and expand docking infrastructure in consistently high-demand locations identified by the model.

Together, these recommendations turn predictive insights into concrete strategies that can enhance profitability, improve rider satisfaction, and strengthen the company's competitive position.

References

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

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 785–794. https://doi.org/10.1145/2939672.2939785

Chen, T., He, T., Benesty, M., Khotilovich, V., Tang, Y., Cho, H., ... & Li, Y. (2023). xgboost: Extreme gradient boosting (R package version 1.7.6) [Computer software]. Comprehensive R Archive Network (CRAN). https://CRAN.R-project.org/package=xgboost

Grolemund, G., & Wickham, H. (2011). lubridate: Make dealing with dates a little easier (R package version 1.9.2) [Computer software]. Comprehensive R Archive Network (CRAN). https://CRAN.R-project.org/package=lubridate

Hastie, T., Tibshirani, R., James, G., & Witten, D. (2021). An introduction to statistical learning: With applications in R (2nd ed.). Springer. https://www.statlearning.com/

Kaplan, J. (2022). fastDummies: Fast creation of dummy (binary) columns and rows from categorical variables (R package version 1.7.3) [Computer software]. Comprehensive R Archive Network (CRAN). https://CRAN.R-project.org/package=fastDummies

Liaw, A., & Wiener, M. (2002). Classification and regression by randomForest. R News, 2(3), 18–22. https://CRAN.R-project.org/package=randomForest

Mavrodiev, H. (2017). London bike sharing dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/hmavrodiev/london-bike-sharing-dataset

Neuwirth, E. (2022). RColorBrewer: ColorBrewer palettes (R package version 1.1-3) [Computer software]. Comprehensive R Archive Network (CRAN). https://CRAN.R-project.org/package=RColorBrewer

OpenAI. (2025). ChatGPT (February 2025 version) [Large language model]. https://openai.com/chatgpt

Seber, G. A. F., & Lee, A. J. (2012). Linear regression analysis (2nd ed.). Wiley. https://doi.org/10.1002/9780471722199

Therneau, T., & Atkinson, E. (2023). rpart: Recursive partitioning and regression trees (R package version 4.1.23) [Computer software]. Comprehensive R Archive Network (CRAN). https://CRAN.R-project.org/package=rpart

Wickham, H., & Kuhn, M. (2023). tidyverse: Easily install and load the tidyverse (R package version 2.0.0) [Computer software]. Comprehensive R Archive Network (CRAN). https://CRAN.R-project.org/package=tidyverse

Appendix

Metadata

Variable	Description
`timestamp`	Date and time of the observation (used to extract hour, weekday, etc.)
`cnt`	Total number of bikes rented during the hour *(Target variable)*
`t1`	Actual temperature in degrees Celsius
`t2`	Feels-like temperature in degrees Celsius
`hum`	Humidity as a percentage
`wind_speed`	Wind speed in kilometers per hour
`weather_code`	Categorical weather condition (see codes below)

Variable	Description
`is_holiday`	Binary indicator: 1 if the day is a holiday, 0 otherwise
`is_weekend`	Binary indicator: 1 if the day is a weekend (Saturday or Sunday), 0 otherwise
`season`	Meteorological season: 0 = Spring, 1 = Summer, 2 = Fall, 3 = Winter

Weather Code Categories

- `1` = Clear (may include haze/fog/patches of fog)
- `2` = Scattered clouds / few clouds
- `3` = Broken clouds
- `4` = Cloudy
- `7` = Rain / light rain shower / light rain
- `10` = Rain with thunderstorm
- `26` = Snowfall
- `94` = Freezing fog