Project Scoping - Pedal Pulse: Predicting Bluebike Demand

Team Members : Group 1

- **Dhruv Miyani**
- **Divya Hegde**
- **Ritika Pankaj Dhall**
- **Muskan Khandelwal**
- **Sai Kashyap Cheruku**
- **Shreyashri Vishwanath Athani**

1. Introduction

Bluebikes is the regional bike-sharing system serving the Boston metropolitan area, offering access to over 4,000 bicycles, including e-bikes, and more than 400 docking stations. The system integrates with multiple transit options, with stations strategically located near other Boston public transit stations. To ensure optimal availability across the network, Bluebikes employs a fleet of 4-5 rebalancing vans, for continuous redistribution based on real-time data.

As the Bluebikes system has expanded significantly over the past decade, several operational challenges have emerged, including seasonal fluctuations in demand, varying usage patterns across neighborhoods, and the need for efficient redistribution. These challenges have highlighted the importance of determining station-level demand and addressing the demand-supply imbalance. This project focuses on forecasting station-level demand to improve the system's ability to balance bicycle distribution. By addressing the current challenges, this initiative aims to reduce operational inefficiencies, lower costs, and ultimately increase profitability. Enhanced availability of bikes, driven by improved supply management, is also expected to boost customer satisfaction and overall system utilization.

2. Dataset Information

2.1 Dataset Introduction

The dataset comprises comprehensive trip histories & station data from the Bluebikes bicycle-sharing system in the Boston area. The purpose of this dataset is to predict bike demand at specific times by analyzing factors like time, day, weather, and events.

2.2 Data Description

- **Dataset:** Bluebikes Comprehensive Trip Histories
- **Collected from:** January 2011 to August 2024 (Monthly Trip Data in CSV format)
- **Size:** Our final dataset is approximately 887 MB. It has 2,766,409 records. We have combined monthly data from February to September, with varying sizes of 234,000 to 500,000 records each month. The final dataset has weather data combined with it.

- **Data features:** The data is combined with weather data, with 31 features in total.
- **Data Labels Generation:** The target variable y is derived by aggregating the total number of bikes undocked (`BikeUndocked`) at each station for every hour (`started_at`) through a `resample('H')` operation. This preprocessing step is performed within Vertex AI as part of the model development pipeline.
- **Label Relevance:** The generated labels (y) serve as the key target variable for forecasting tasks, capturing hourly bike usage patterns at each station. These labels are constructed from historical data, ensuring they accurately reflect past trends and operational metrics to support reliable predictions.

2.3 Data Types

- **Trip Histories:** Numerical (trip duration, bike ID), Categorical (user type, gender), Time (start and stop times), Geospatial (start/end station IDs and names)
- **Station Data:** Numerical (station ID, total docks), Geospatial (latitude, longitude), Categorical (municipality, station name)

2.4 Data Sources

- Trip Histories: Downloadable files of Bluebikes trip data, updated quarterly
- Station Data: Downloadable station data
- Weather Underground: Weather data is scraped from the site https://www.wunderground.com/.

2.5 Data Rights and Privacy

- Governed by a license agreement with Lyft Bikes and Scooters, LLC.
- Data can be used for analysis, development, and non-commercial purposes.
- Data is anonymized and does not contain personally identifiable information (PII).
- Compliant with local data protection regulations like GDPR.

3. Data Planning and Splits

3.1 Loading

The data is available as CSV files, which can be loaded using `pandas` or similar data manipulation libraries. Real-time station data can be accessed via API calls.

3.2 Preprocessing

- Cleaning: Remove incomplete or erroneous records, such as trips shorter than 60 seconds.
- Feature Engineering: Convert time data (start/stop) to appropriate formats for temporal analysis (e.g., hour of the day, day of the week).
- Data Transformation: Scale numerical features and prepare datasets for machine learning models.

3.3 Managing Data

Memory-efficient operations (e.g., chunk loading for large files) might be necessary. For real-time data, storing snapshots of station availability can support predictive models.

3.4 Splitting Strategies

Temporal Splitting: Use a time-based split to separate historical data for training and more recent data for testing. Cross-validation: Perform k-fold cross-validation to ensure model robustness.

4. GitHub Repository

GitHub

```
bluebikes-demand-prediction/

 README.md
 data/
    raw/
    processed/
 notebooks/
    data_exploration.ipynb
 src/
    app.py
 models/
 tests/
 airflow/
    dags/
 ci/
    github-actions.yml
 requirements.txt
```

5. Project Scope

5.1 Problems

Bike-sharing systems, including Bluebikes, often struggle with accurately forecasting station-level demand, which leads to imbalances where some stations are overstocked while others run out of bikes. This leads to user frustration and operational inefficiencies due to the need for frequent, unplanned rebalancing trips, increasing costs. Time-sensitive demand variations, such as during rush hours or special events, make it even harder to maintain balance across the network. While real-time data is available for monitoring, the lack of dynamic prediction models keeps demand forecasting reactive and inefficient, limiting the system's ability to prevent these imbalances proactively.

5.2 Current Solutions

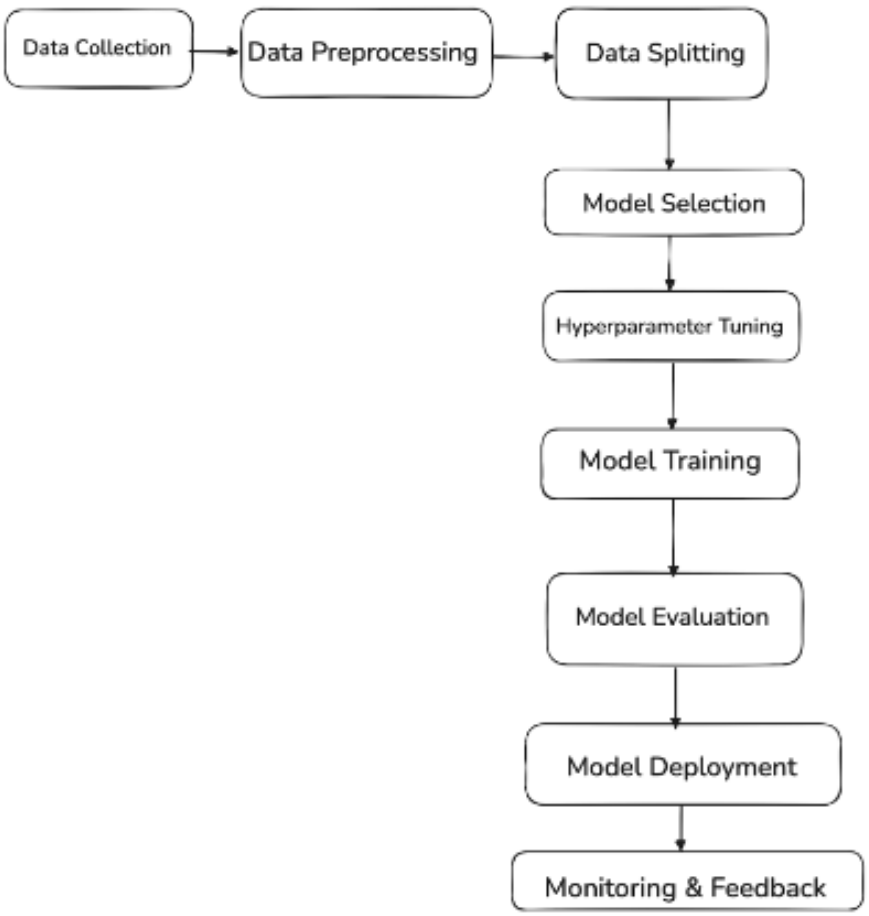
Many bike-sharing systems rely on simple threshold-based systems to trigger bike redistribution efforts when stations hit critical levels of emptiness or overcrowding, aiming to manage demand effectively. Workers manually transport bikes between stations based on real-time data, but this approach is resource-heavy as unplanned trips can drive up operational costs. Some systems employ linear regression models to predict short-term demand but struggle to account for complex, nonlinear factors that affect usage patterns.

5.3 Proposed Solutions

- Advanced demand forecasting: Develop predictive models on historical and current data along with external factors like weather and events. These models will forecast bike and e-bike demands at different stations, allowing for proactive decision-making.
- Time Series Forecasting: Utilize machine learning models to capture temporal patterns and predict station occupancy and bike demand for future time intervals.

- Automated MLOps Pipeline: Implement a continuous MLOps pipeline to retrain models as new data comes in, ensuring forecasting remains accurate and responsive to changing trends.

6. Current Approach Flow Chart and Bottleneck Detection



Data Collection

Bottleneck: Limited or inconsistent data, real-time data delays.
Solution: Automate data collection using APIs for real-time weather and bike data. Implement data validation and quality checks to ensure data consistency.

Data Preprocessing

Bottleneck: Complex preprocessing, high computational cost.
Solution: Use automated pipelines (e.g., Scikit-learn Pipelines) for preprocessing. Reduce dimensionality early with techniques like PCA to decrease computation.

Data Splitting

Bottleneck: Risk of imbalanced splits and overfitting.
Solution: Use stratified splitting or time-series split methods to ensure balanced and meaningful datasets.

Model Selection

Bottleneck: Time-consuming experimentation without automated tools.
Solution: Use automated model selection tools such as MLFlow to reduce manual experimentation time.

Hyperparameter Tuning

Bottleneck: Computationally expensive and time-consuming.

Model Training

Bottleneck: Long training times for complex models or large datasets.
Solution: Implement early stopping to avoid overfitting. Use distributed training (e.g., PyTorch Lightning) for large datasets.

Model Evaluation

Bottleneck: High evaluation cost with multiple model versions.
Solution: Use efficient cross-validation techniques and log results using tools like MLFlow for quick comparison and tracking.

Model Deployment

Bottleneck: Integration with production and managing scalability.
Solution: Automate CI/CD pipelines using tools like GitHub Actions, and deploy with Docker/Kubernetes for scalability.

Monitoring and Feedback

Bottleneck: Model drift and performance degradation over time.
Solution: Use real-time monitoring tools like Prometheus and Grafana to detect data drift early, and set up automatic retraining pipelines with tools like Airflow/Kubeflow.

7. Metrics, Objectives, and Business Goals

Key Metrics to Evaluate Project Success

To provide performance and usability insights of the forecasting model, traditional regression metrics such as Root Mean Squared Error(RMSE) and demand planning metrics like Forecast Error, Weighted Mean Absolute Percentage Error(WMAPE) and Forecast Bias may be chosen.

- RMSE and Forecast Error metrics offer absolute (sign-agnostic) and magnitude-based interpretations of model performance, respectively.
- WMAPE is used for a more nuanced forecast accuracy understanding, by giving more weight to higher-demand stations, which are usually significant in deciding user experience.

- Tracking bias in the forecast error helps to identify trends of underestimation/overestimation of demand thereby preventing prolonged skewing in the forecasts.

Project Objectives

Bluebikes is committed to providing affordable and reliable public transportation to all residents of the Metro Boston Area. Apart from flexible subscription options, the bike-sharing service also runs an income-eligible program that offers subsidized membership to underrepresented and underserved communities. Sustaining an operation riddled with challenges of fluctuating demand and maintaining rider satisfaction can incur a surge in operational costs and resources.

Our project aims to provide a robust service for demand forecasting. We believe that our efforts will succeed in alleviating these pain points and enable a public-focused organization like Bluebikes to conduct and scale its operation, without facing supply-demand imbalances. This would lead to higher customer satisfaction and greater subscriber loyalty, ultimately increasing revenue and supporting the scaling of the ride-sharing service along with its community outreach programs.

8. Failure Analysis

- **Data:** Inaccurate and inconsistent trip data, along with the unavailability of historical data for newly set-up stations, may hinder forecast performance. Combining datasets from different sources (e.g., trip-specific data with local weather data) can complicate the capture of seasonality and trends. Additionally, low-activity stations may be underrepresented in the dataset, necessitating a careful sampling and splitting strategy to ensure adequate training and validation for the model.
- **Model Selection and Training:** Identifying a model capable of capturing ever-evolving demand patterns may lead to the selection of complex models (e.g., neural networks), which can lack interpretability. Striking a balance between underfitting and overfitting is crucial, especially given the inherently noisy training data. Finding optimal hyperparameters can be time-consuming and computationally expensive, depending on the model's complexity.
- **Model Evaluation:** Evaluating model performance across various metrics may yield conflicting results, leading to uncertainty in understanding forecast efficacy.
- **Infrastructure Scalability:** Inefficient scaling of the chosen MLOps platforms can hinder the ability to handle increased data ingestion and training. Without proper scaling, issues such as processing delays, higher costs, and degraded performance during peak usage may arise.

Pipeline Failures and Mitigation Strategies

- **Ingestion Failures:** API downtime or changes in request schemas can lead to data retrieval issues.
  - **Mitigation:** Implement robust error handling and schema validation checks to accommodate format changes.
- **Deployment Issues:** Incompatibility between development and production environments can cause runtime failures or unexpected behaviors. Errors in model serialization and deserialization may result in incorrect predictions and increased downtime.
  - **Mitigation:** Use containerization to ensure consistency, along with proper model version control and a model registry.
- **Monitoring and Alerting Failures:** Delays or missed alerts for critical issues can arise from misconfigured thresholds or non-coverage of essential components or metrics.
  - **Mitigation:** Implement log aggregation and centralized monitoring while regularly tuning alert thresholds.

9. Deployment Infrastructure

9.1 Data Ingestion and Preprocessing

Data from the BlueBikes platform (station data, historical trips) will be ingested into cloud storage (Google Cloud Storage).
Apache Airflow for orchestrating scheduled jobs to fetch BlueBikes data.
GCP Storage (GCS) or Amazon S3 for raw and processed data

9.2 Model Training and Experimentation

GCP AI Platform/SageMaker for large-scale distributed training. Periodically retrain models as new trip data is ingested into the pipeline.
Jupyter Notebooks for interactive exploration and initial experimentation.
DVC for tracking dataset and model versions.
Google Cloud Platform Provides scalable cloud infrastructure for hosting, managing, and deploying machine learning models and applications.

9.3 Model Registry

MLflow for model tracking, versioning, and registry. It serves as a centralized location where all model versions are logged and can be retrieved as needed.

9.4 Model Deployment and Serving

FastAPI: Build a REST API to expose the model as a service.
Docker, Kubernetes: The API will be containerized using Docker for consistency and deployed on Kubernetes or GCP Cloud Run for scalable serving. This ensures that the model is accessible in a production environment for real-time predictions.

9.5 Monitoring and Logging

Prometheus & Grafana: Set up Prometheus to monitor model performance, request rates, and server health. Use Grafana for real-time dashboards and alerting.

9.6 Continuous Integration/Continuous Deployment (CI/CD)

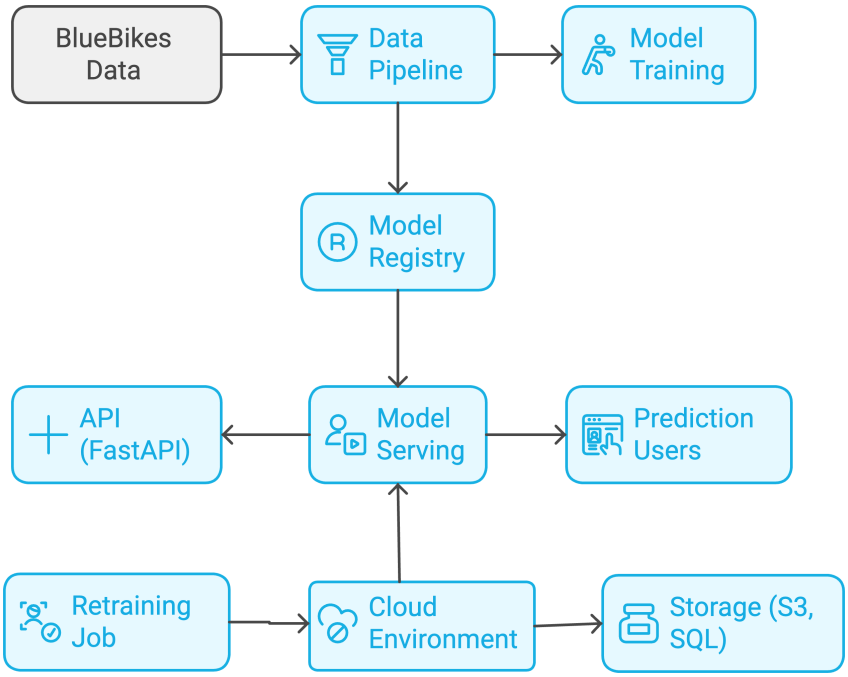
GitHub Actions: Automate testing, linting, and deployment for the model pipeline. Integrate DVC into GitHub Actions to automatically retrain models when new data is added. Use GitHub Actions to deploy the model and API to the appropriate cloud services (GKE, Cloud Run).

9.7 Retraining Pipeline

Scheduled retraining jobs will be managed through **Apache Airflow**. The retraining process will be automatically triggered based on data drift or time intervals, ensuring the model remains accurate as new data is ingested. Airflow DAGs will automate the entire retraining process, from preprocessing to model deployment, keeping the system up-to-date with minimal manual intervention.

9.8 Storage (Data & Model Artifacts)

All raw and processed data, as well as model artifacts, will be stored in **Google Cloud Storage (GCS)** or **Amazon S3**. This storage solution provides scalability and reliability for managing large datasets and models, ensuring that all artifacts are securely stored and easily accessible for future use.



10. Monitoring Plan

The monitoring plan for our MLOps project encompasses five key areas:

10.1 Data and API Monitoring

Focuses on tracking API call responses, data processing time, and API request rates and error rates.

10.2 Model Serving, Training, and Versioning

Covers monitoring inference time and throughput, CPU usage and training duration, loss and accuracy during training, as well as recording model versions, datasets, and hyperparameters. It also includes comparing performance metrics between versions.

10.3 Data Drift and Retraining

Involves implementing data drift detection mechanisms and setting thresholds for automatic retraining triggers.

10.4 Error and Exception Monitoring

Includes implementing structured logging across the pipeline, setting up real-time alerting for critical failures, and establishing an incident response plan.

10.5 Cloud Environment Monitoring

Tracks compute and storage resource utilization, including CPU/GPU usage, disk space, and memory allocation. Monitors cloud costs and ensures service availability and uptime to avoid downtime in model deployment and prediction serving.

11. Success and Acceptance Criteria

11.1 Success Criteria

- **Data Preprocessing and Transformation:** Ensure cleaned data and a functional data pipeline.
- **Accurate Demand Forecasting:** Models should achieve the best accuracy in forecasting bike demand, leading to reduction in station imbalances. Successfully deploy models to production that meet performance benchmarks.
- **Deployment Strategy:** Define a deployment strategy with a functional CI/CD pipeline.
- **Real-Time Decision-Making:** The system must provide accurate demand forecasts in advance to enable proactive rebalancing.
- **User Feedback:** Gather positive user feedback on improvements.

11.2 Acceptance Criteria

- **Clear Objectives:** Establish clear objectives, scope, and project plan.
- **Data Quality:** Validate data quality and pipeline functionality.
- **Model Performance:** Model should achieve minimum accuracy in forecasting bike demand.
- **Scalability of MLOps Pipeline:** The MLOps pipeline should ensure continuous retraining of models without accuracy loss, maintaining or improving predictive performance. Implement monitoring, maintenance, and retraining pipelines based on user feedback and model improvements.
- **Monitoring Systems:** Ensure monitoring systems and maintenance procedures are in place.
- **Documentation:** Provide final documentation and obtain formal sign-off.

11.3 General Criteria

- **Timely Completion:** Complete the project by Dec 8.
- **Compliance:** Ensure compliance with regulations and best practices.

12. Timeline Planning

- **Week 1 (Sept 23 - Sept 29):** Project Kickoff and Planning.
  - Conduct a project kickoff meeting to align the team.
  - Define project objectives, scope.
  - Develop and finalize the initial project plan and timeline.
  - Assign roles and responsibilities to the team.
- **Week 2 (Sept 30 - Oct 6):** Data Preparation and Pipeline Setup.
  - Gather and explore the initial dataset from BlueBikes.
  - Set up Google Cloud Storage (GCS) or Amazon S3 for data storage.
  - Implement the Apache Airflow data ingestion pipeline to fetch and store data.
  - Validate that data ingestion is working as expected.
- **Week 3 (Oct 7 - Oct 13):** Model Development and Validation.
  - Develop and train baseline models using PyTorch or TensorFlow.

  - Perform hyperparameter tuning to optimize the baseline models.
  - Begin tracking experiments with DVC and MLflow for version control.
  - Assess initial model performance and make necessary adjustments.
- **Week 4 (Oct 14 - Oct 20):** Refactoring Codebase and CI/CD Pipeline Setup.
  - Refactor the codebase to ensure modularity and scalability.
  - Define deployment requirements and create the CI/CD pipeline using GitHub Actions.
  - Write unit and integration test cases for model and pipeline validation.
  - Set up continuous integration to automate testing and deployments.
- **Week 5 (Oct 21 - Oct 27):** Containerization and Airflow DAGs Creation.
  - Containerize the environment using Docker for consistent deployment.
  - Create Airflow DAGs for orchestrating data ingestion and model training workflows.
  - Validate containerized environments and DAGs for seamless execution.
- **Week 6 (Oct 28 - Nov 3):** Monitoring, Retraining Pipeline and Backend Coding.
  - Develop the API using FastAPI to expose the model as a service.
  - Implement data mappers to ensure smooth integration with external systems.
  - Build the retraining pipeline with Airflow to automatically trigger retraining based on new data or performance drift.
- **Week 7 (Nov 4 - Nov 10):** Cloud Deployment and Maintenance Setup.
  - Deploy the model and API to the cloud using Kubernetes or GCP Cloud Run.
  - Set up Prometheus and Grafana for real-time model monitoring and health checks.
  - Develop a simple user interface to interact with the deployed model.
  - Ensure automatic logging and monitoring are in place for post-deployment maintenance.
- **Week 8 (Nov 11 - Nov 17):** Project Testing and Optimization.
  - Conduct end-to-end testing of the pipeline, model, and deployment.
  - Run performance and load tests to optimize API and model performance.
  - Make final optimizations based on testing results to ensure stability in production.
- **Week 9 (Nov 18 - Nov 24):** Documentation and Finalization.
  - Finalize all technical documentation, including the pipeline, model, and deployment strategies.
  - Document all integrations, dependencies, and troubleshooting guides for future maintenance.
  - Ensure version control of documentation is up-to-date.
- **Week 10 (Nov 25 - Dec 1):** Project Review and Handover.
  - Review the entire project and the pipeline in it and finalize the project.
  - Complete presentation slides and prepare for the MLOps Expo.
- **Week 11 (Dec 2 - Dec 8):** Project Closure.
  - Conduct a final project review.
  - Present project outcomes, including performance metrics and lessons learned.
  - Officially close the project and transition to the maintenance phase.

13. Additional Information

- **Cost and Resource Estimation:**
  - **Google Cloud Storage:** $0.02/month for storing preprocessed data and model artifacts.
  - **Cloud SQL:** $5.00/month for storing MLflow metadata to track accuracy metrics and experiments.
  - **Vertex AI Workbench:** $10.00/month for preprocessing and exploration of 200 MB of data.
  - **Vertex AI Training:** $4.00/month for training models on 200 MB of combined data (used for 11 months).
  - **Model Deployment:** $50.00/month for serving models trained on 75 MB of processed data.
  - **Monitoring and Logging:** $30.00/month for logging preprocessing steps, model metrics, and selecting the best-performing model.
  - **Total Monthly Estimate:** $100/month.
- **Potential Risks and Their Impact:**
  - **Ingestion Failures:**
    - **Impact:** API downtime or schema changes can lead to missing or delayed real-time data, resulting in incorrect bike availability predictions. Users may face empty or overcrowded docking stations.
  - **Data Quality Issues:**
    - **Impact:** Inconsistent or inaccurate trip data can skew forecasts, leading to suboptimal bike redistribution. Users may experience inconvenience due to poor station management.
  - **Preprocessing and Feature Engineering Failures:**
    - **Impact:** Incorrect or incomplete preprocessing can cause flawed predictions, leading to inefficiencies in managing peak-demand periods, frustrating users due to unavailability of bikes.
  - **Model Selection and Training Errors:**
    - **Impact:** Choosing overly complex or misconfigured models can cause delays in predictions or inaccurate forecasts, leading to frequent imbalances and dissatisfied users.
  - **Model Deployment Failures:**
    - **Impact:** Incompatibilities between development and production environments can result in downtime, leaving users without reliable bike availability during critical periods.
  - **Monitoring and Alerting Failures:**
    - **Impact:** Missed alerts on performance degradation or pipeline issues can allow errors to persist, leading to prolonged supply-demand mismatches and decreased user satisfaction.
  - **Scaling and Infrastructure Issues:**
    - **Impact:** Inefficient scaling during high-demand periods can slow predictions or cause the system to fail, directly affecting bike availability and user trust.