**Status Report 2:** **Advancement and Future Actions for the Bike-Sharing System Section**

**Section 1: Team Member Roles**

**Data engineering project lead- Vamshi Krishna Perabathula**

Oversaw first Postgres Onal and Kafka integration as well as Docker environment building.

Oversaw real-time data streaming configurations.

**Model development- Sai Niharika Hari**

Helped create Docker containers for data processing and consumption.

coordinated feature extraction and preprocessing programs for real-time simulation data.

**Section 2:** **Objective of Projects and Development**

**Project Goals Recap:** Our aim is to use predictive analytics to forecast bike demand across various stations. This will allow bike-sharing companies to better manage resources and ensure bikes are available where they’re most needed.

**Objectives:**

**General Overview:**

* Machine learning allows one to forecast bike demand across stations.
* Make sure bikes are in highly sought-after sites to maximize bike-sharing programs.

**Development Since Last Report:**

* System Components Arranged
* Successfully produced Docker containers for: configured to consume simulated bike rental data in real time.
* Postgres: Designed and operational is the database structure.
* Spark: Data cleansing and processing first set-up completely.

**Real-time data simulation**

* Designed and tested to replicate real bike rental activities is a Python software.
* Bike topic is a Kafka subject receiving data streams into.

**Database Settings:**

* Postgres is housed in a Docker container.
* Data—including station, timestamp, and journey length fields—has been stored using a schema.
* Using consumers, Postgres is effectively absorbing data from Kafka.

**Cleansing and processing data:**

* Preprocessing streaming data with Apache Spark addresses missing or erroneous items.
* Outliers in station use and travel times.
* Postgres is storing aggregated data including hourly bike demand per station for training.

**Engineering features:**

Features taken from hourly demand include:

* Popular patterns particular to a station.
* Patterns in time of day and day-of-week.

**Initial model training:**

* A rudimentary XGBoost model has been developed from a subset of past events.
* Early assessment measures show encouraging outcomes; further adjustment is needed.

**Development of Flask APIs:**

* There is now a rudimentary Flask API designed to fulfil predictions.
* Input characteristics and return anticipated demand have been accepted using endpoint /predict.

**Imagining:**

* Tableau is related to PostgreSQL.
* First dashboards reveal real-time bike demand patterns.
* Station-wise historical consumption trends.

**Section 3: Difficulties and Problems Seen**

**Synchronized Real Time:**

* Although Kafka streams are consistent, managing asynchronous data arrival from many sources calls for improvement.

**Complexity in Feature Engineering:**

* Including outside data sources like events and weather calls for further preprocessing logic.

**Issues with scalability:**

* Optimizing the coordination across Kafka, Spark, and PostSQL containers becomes more important as the dataset increases.

**Refining and accuracy of models:**

* Particularly for demand spike management, early XGBoost model forecasts call for improvement.

**Section 4: upcoming actions**

**Complete pipelines of data pipes:**

* Improve data integrity and smooth intake by streamlining the real-time pipeline from Kafka to Postgres.

**Advanced features engineering:**

* Combine outside data—such as public events or weather—to raise model performance.

**Model tuning entails:**

* Using cross-valuation, hyperparameter adjustment on the XGBoost model increases accuracy.
* Examine group approaches for improved forecasts.

**Apartment Deployment:**

* Add batch predictions for every station to extend Flask API capability.

**Visualization Improvement Strategies:**

* Add demand comparisons between actual and projected overlays to dashboards.

**Complete Systems Testing:**

* Use real-time simulation to do an end-to- end test looking for bottlenecks.