Waiting Time Prediction Challenge

## Introduction

Efficiently predicting service request waiting times is essential for improving operations and customer satisfaction. This challenge involves using historical time-stamped transaction data to build a machine-learning model that predicts waiting times. Additionally, Explainable AI (XAI) techniques have been applied to ensure transparency and provide insights into the model’s decision-making process, fostering trust and an actionable understanding of the predictions.

## Data Processing

1. Removing garbage values (“\N”).
2. **Waiting time** calculation in seconds by subtracting start time (SDST) from arrival time (TKIS\_TIME).
3. Perform Label Encoding on **Service Names.**
4. Extract Features: **arrival day, month,** and **year** from arrival time (TKIS\_TIME).
5. Perform conversion **arrival time** (TKIS\_TIME) into seconds.
6. Perform normalization on features.

Five features were used to predict **waiting time**:

1. Arrival Time (seconds)
2. Arrival Day (day in number)
3. Arrival Month (month in number)
4. Arrival Year
5. Encoded Service Name

## Data Visualization

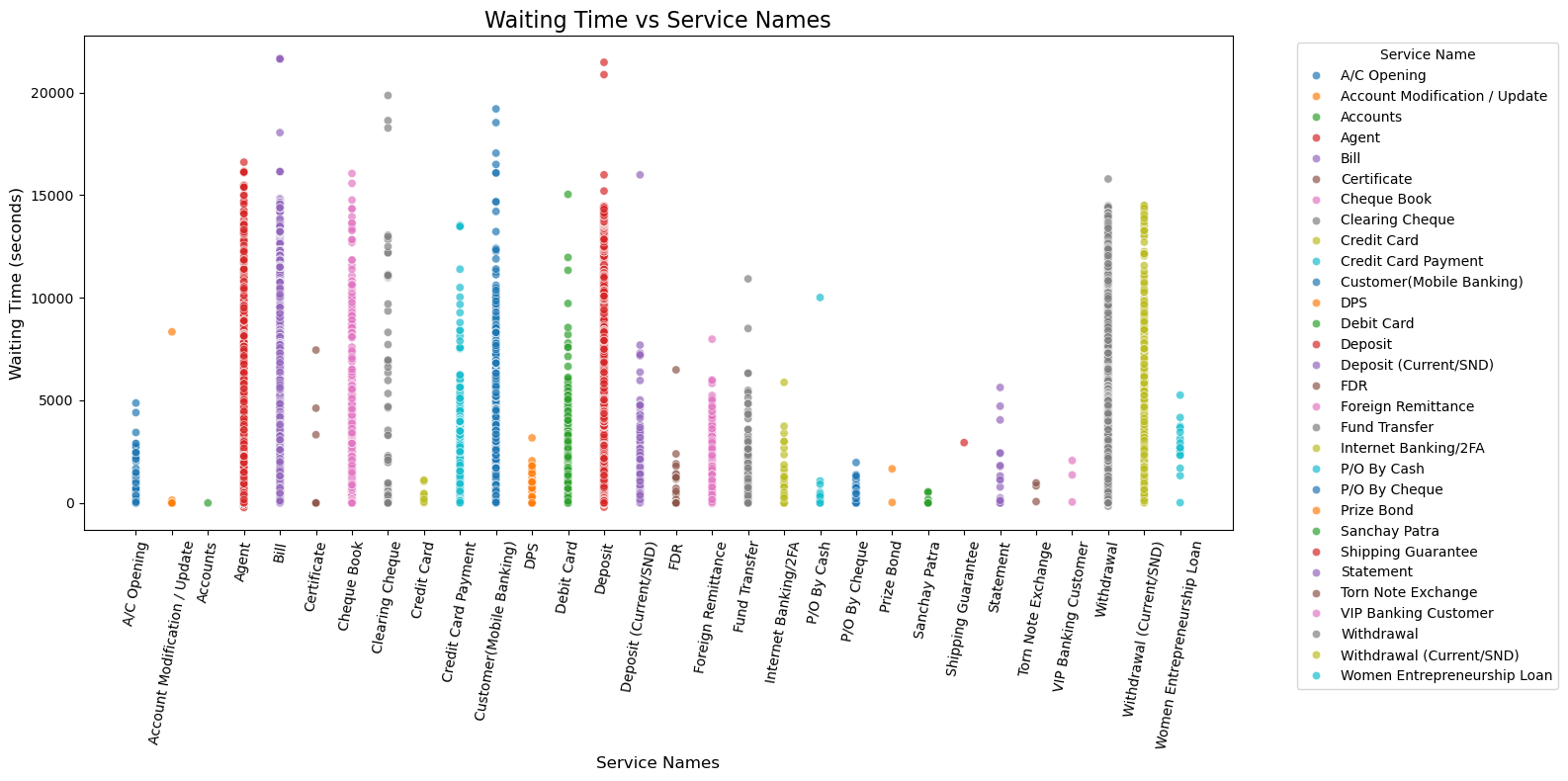


Figure 1: Waiting Time vs Service Names.

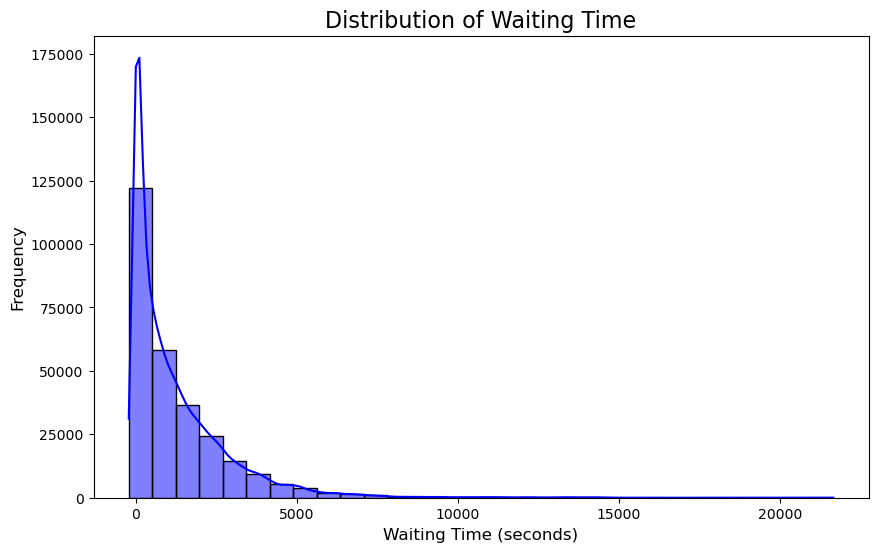


Figure 2: Distribution of waiting time.

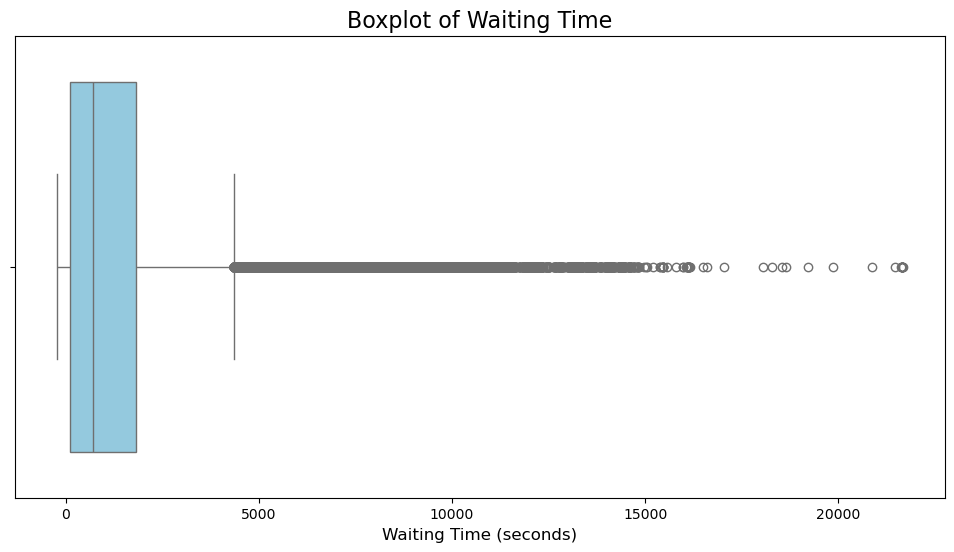


Figure 3: Boxplot to find outliers.

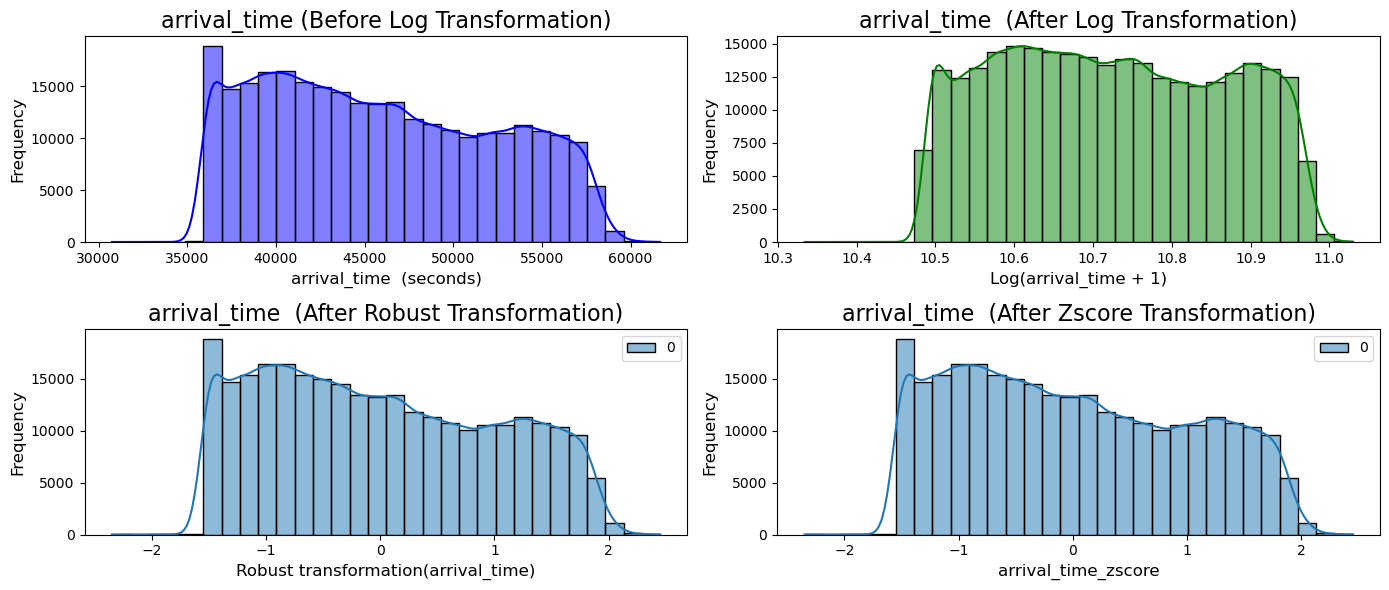


Figure 4: Normalization of waiting time.

## Methodology

After cleaning the data and extracting relevant features, feature normalization was applied. Various regression models were then trained on the normalized dataset to predict waiting times. These models were assessed using evaluation metrics such as R-squared. After comparing their performance, the best-performing model was selected, and Explainable AI (XAI) techniques were integrated into it to enhance interpretability.

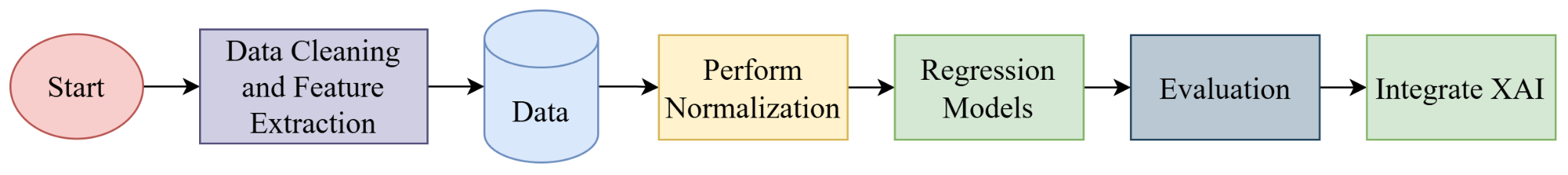


Figure 5: Workflow diagram of the task.

### Evaluated Regression Models

1. Linear Regression
2. Decision Tree Regression
3. Random Forest Regression
4. XGBoost Regression

### Evaluation Metrics

1. R Squared
2. Mean Absolute Error (MAE)
3. Mean Squared Error (MSE)
4. Root Mean Squared Error (RMSE)

### Explainable AI

1. LIME (Local Interpretable Model-agnostic Explanations)
2. SHAP (Shapley Additive explanations)

## Key Challenges

1. Managing a significant number of outliers.
2. Dealing with high variability in the data.
3. Addressing the limited availability of features.
4. Optimizing model performance through hyperparameter tuning.
5. Selecting the most suitable model for the task.

## Results

The models were trained, tested, and evaluated with and without normalization, yielding identical results in both scenarios. Then, hyperparameter tuning was conducted to improve performance.

Table 1: Evaluation result without hyperparameter tuning.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models/Evaluation Metrics | MAE | MSE | RMSE | R Squared |
| LR | 1074..27 | 2418000.59 | 1554.99 | 0.06 |
| DTR | 229.49 | 521303.399 | 722.01 | 0.79 |
| RFR | 220..98 | 344745.21 | 587.15 | 0.86 |
| XGBoost | 718.77 | 1161605.02 | 1077.77 | 0.55 |

Table 2: Evaluation result with hyperparameter tuning.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Models/Evaluation Metrics | MAE | MSE | RMSE | R Squared |
| LR | 1074..27 | 2421484.67 | 1556.11 | 0.06 |
| DTR | 229.49 | 521834.23 | 722.38 | 0.79 |
| RFR | 227.26 | 327097.16 | 571.92 | 0.87 |
| XGBoost | 229.14 | 392439.01 | 626.44 | 0.84 |

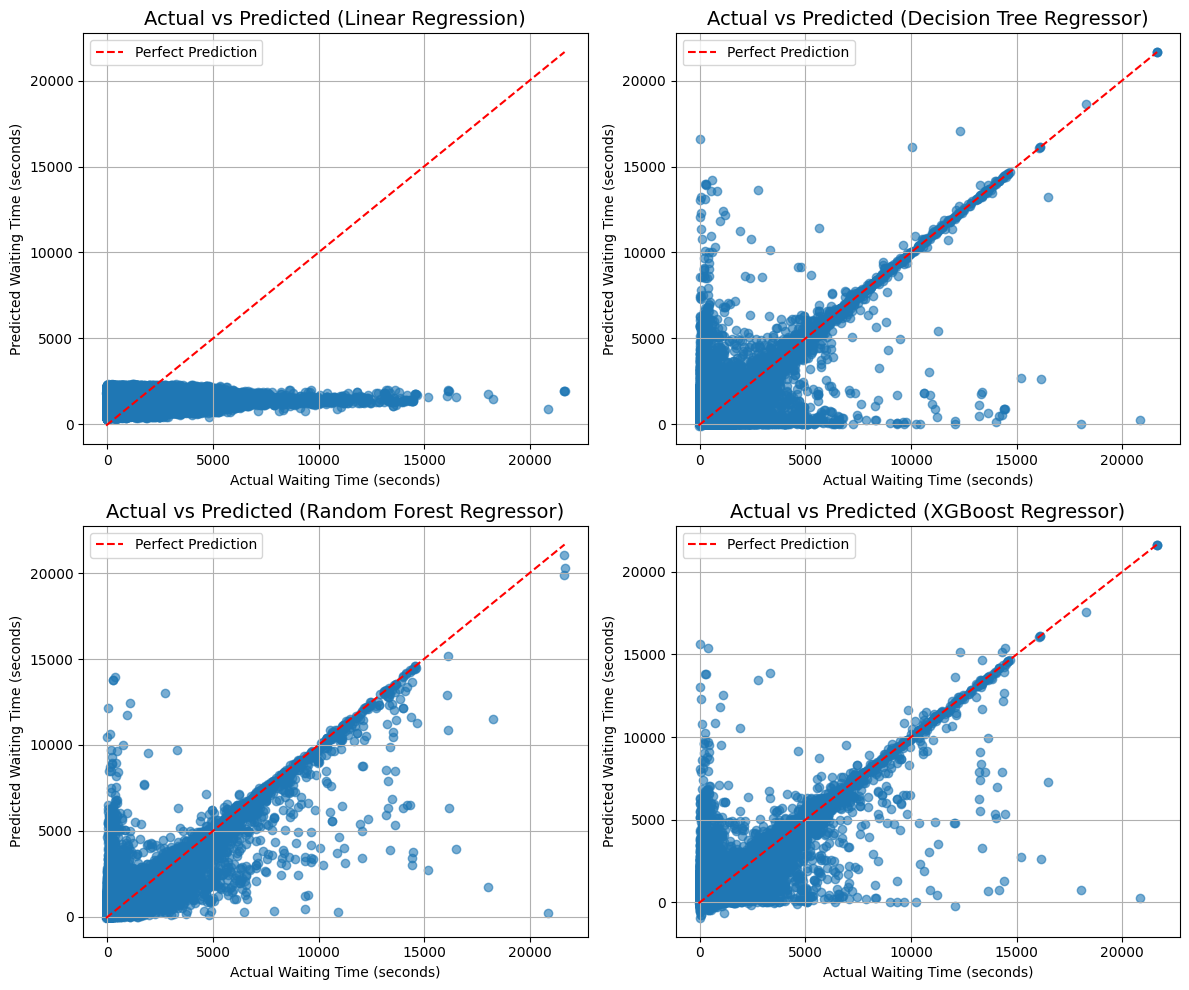


Figure 6: Fitted curves for all models.

## Explainable AI

The Random Forest Regressor (RFR) was the best model in terms of evaluation. Explainable AI techniques like LIME and SHAP were integrated to enhance interpretability and provide insights into feature importance and prediction behavior.

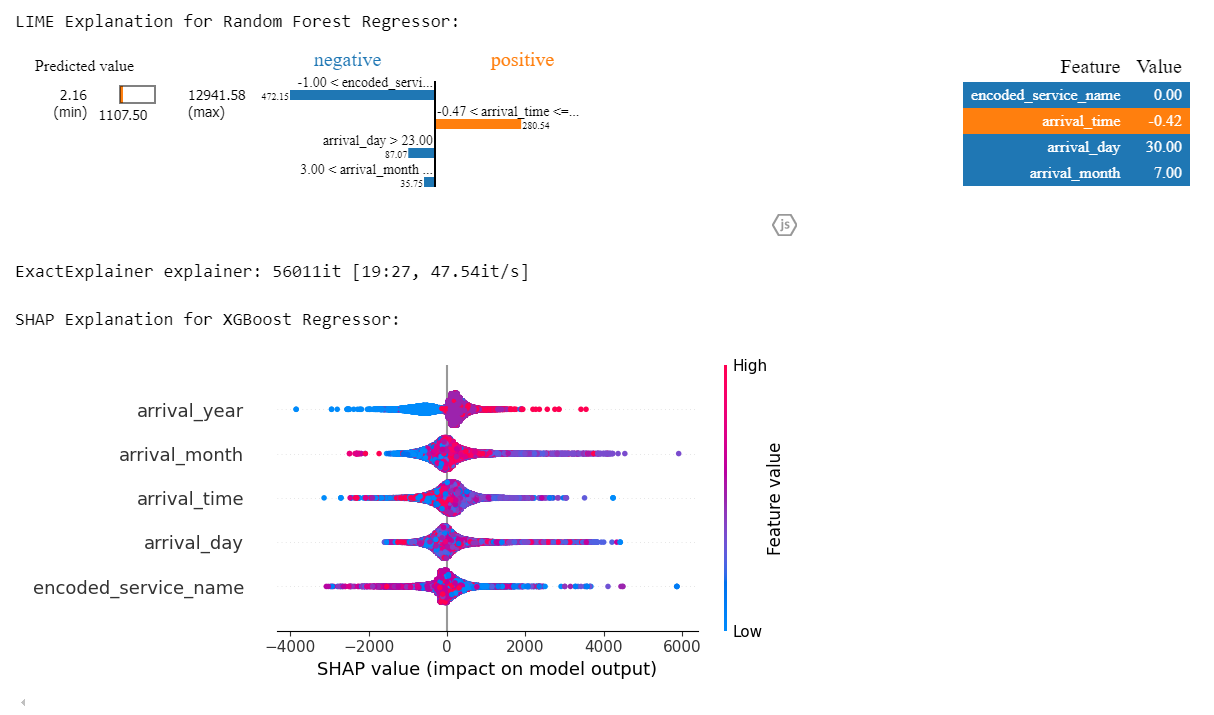


Figure 7: Explainable AI integration.

## Bonus Contribution

### Real-Time Prediction

The best model is successfully tested for real-time waiting time prediction. The system allows users to select a service name, and the model automatically retrieves the current timestamp (arrival time in seconds, day, month, and year) to generate an instant waiting time prediction. The system ensures fast and accurate results by leveraging FastAPI for deployment, Random Forest Regressor for prediction, and label encoding for service names. The optimized pipeline enables efficient handling of real-time user requests while maintaining scalability for larger datasets and higher traffic loads.

### Scalability

To maintain model performance in the ML model, it is essential to update the production model on time; otherwise, it will not perform well. We should create a CI/CD pipeline for the production model to update the model in time. The model must handle large datasets efficiently and support high user traffic to ensure real-time waiting time prediction scalability. Using parallel processing with Spark improves training on large datasets, while FastAPI with Unicorn enables asynchronous API calls for fast predictions. Model caching with Redis reduces redundant computations, and deploying the model on cloud platforms like AWS or GCP ensures auto-scaling based on demand. For faster inference, converting the model to ONNX reduces latency. Continuous learning is managed through automated retraining with Airflow and drift monitoring using AI. Scaling solutions like Kubernetes and MLflow allow seamless model updates and A/B testing before deployment. These optimizations ensure the model remains efficient, responsive, and adaptable to real-world usage.