# **Food Delivery Platform: Part IV Report**

### **Authors:**

### **Name: [Your Name]**

### **NID: [Your NID]**

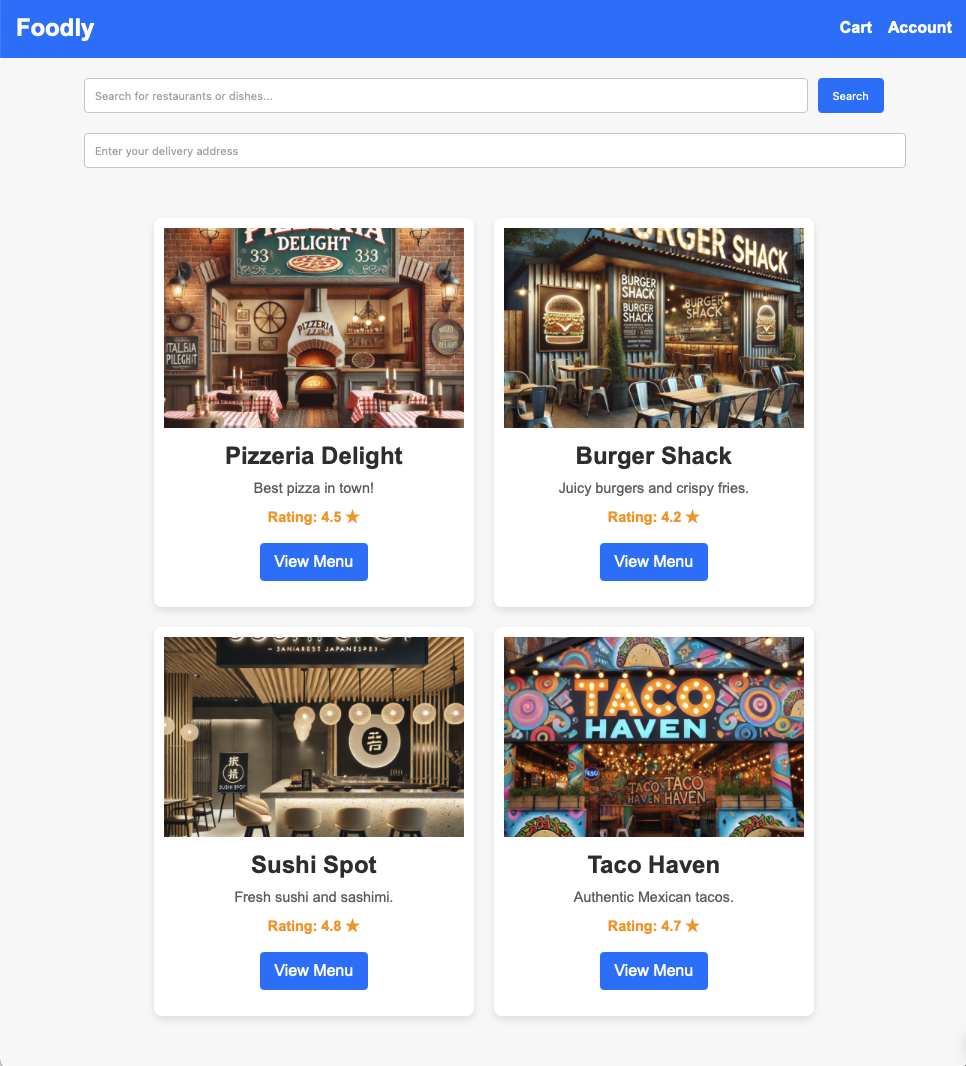
## 

## **Use Case Design**

### Our food delivery platform aims to connect customers with restaurants, enabling a seamless ordering, delivery, and feedback process. Below are the key business use cases for this data-driven, workflow-based platform:

### Home page

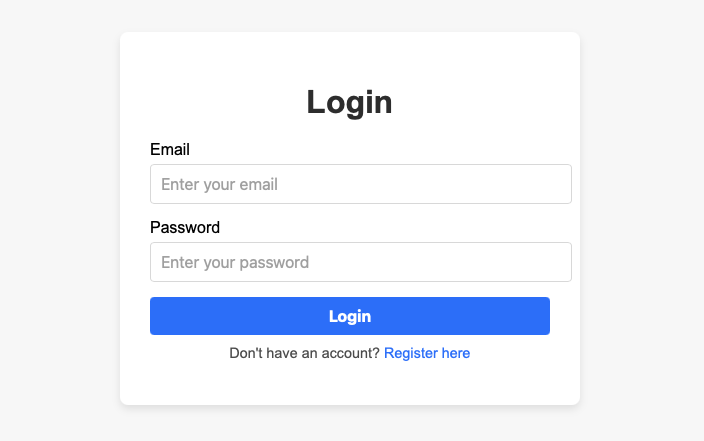
* + Customers can enter their delivery address to see restaurants nearby, search restaurants and view their cart and account information
  + The name, ratings, brief introduction of restaurants are listed

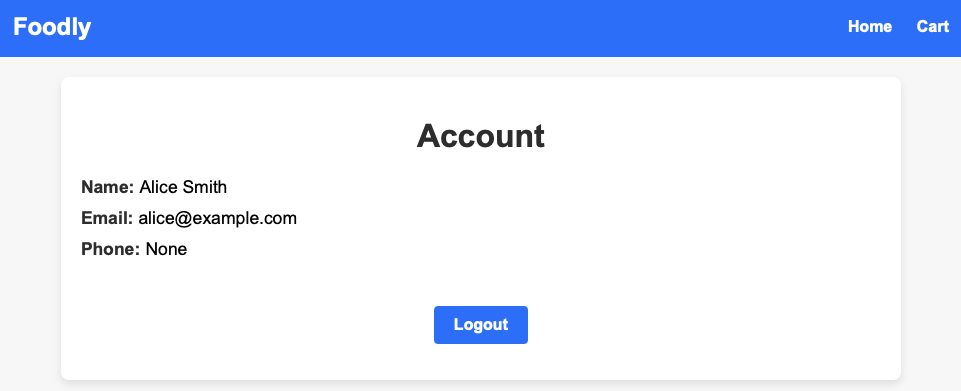


### Customer Registration and Login

### Customers can register and log in to the platform, and view their account information in “Account”.

* + Customer database is updated on a real-time basis to make sure data consistency, accuracy and security.

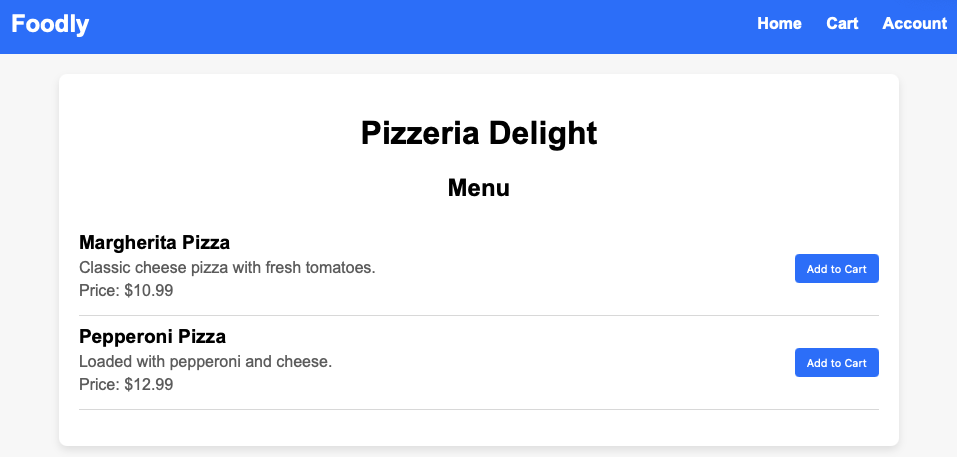




### Restaurant and Menu Browsing

### Customers can browse restaurant listings, view menus and add items to the cart “Cart”

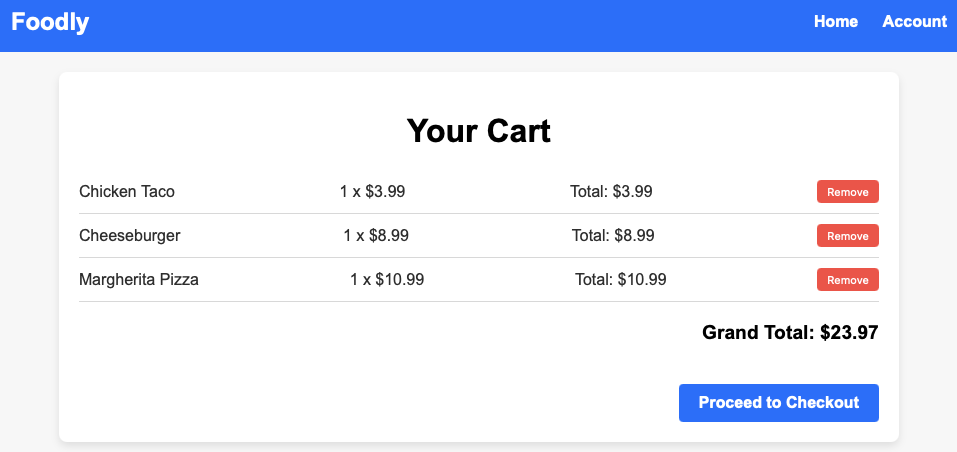
* + At the “Cart” page, customers can proceed to checkout or edit the list of items.

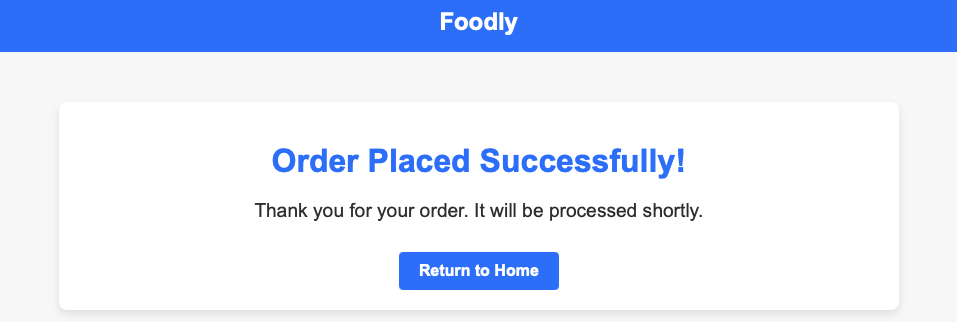


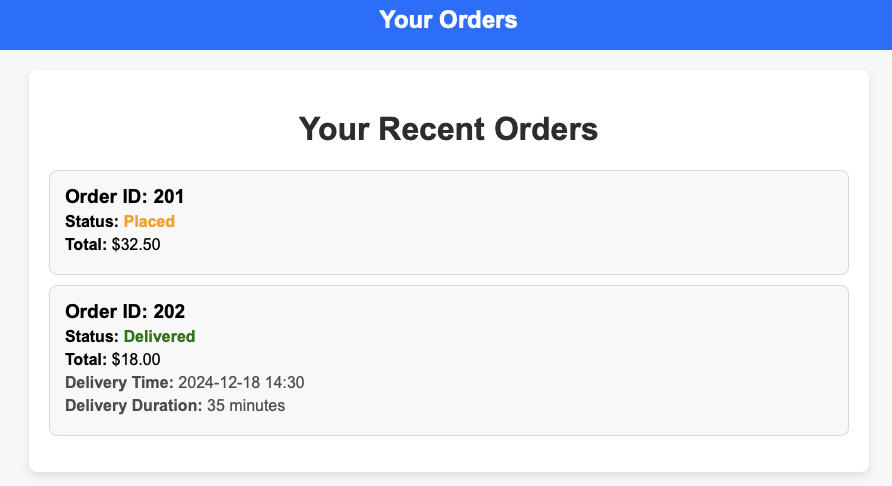
### Order Placement

### Customers can place orders, track order status, and manage past orders.

* + The delivery time of each order is recorded for machine learning analysis and business decision-making

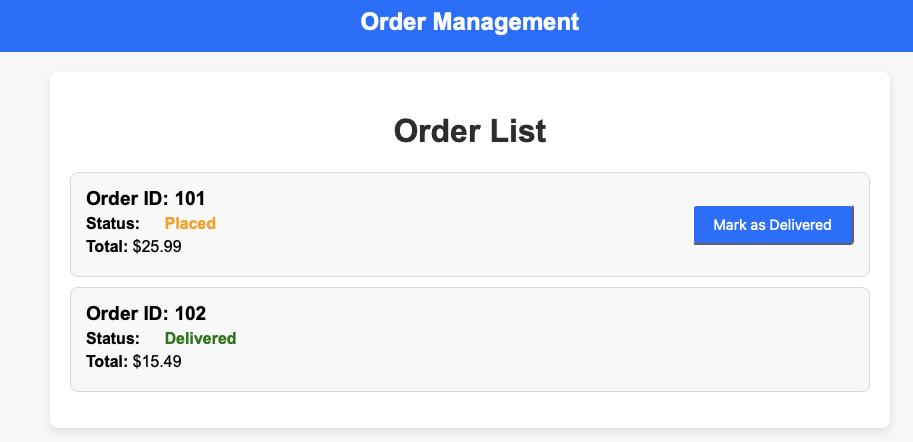






### Delivery Management

### Delivery personnel can view and update delivery statuses. They click the “Mark as Delivered” when an order is delivered.



### Promotion and Discounts

### Restaurants can offer promotions, and customers can apply discount codes at checkout.

## **Implementation**

We developed the food delivery platform using **Flask** for the backend and **HTML/CSS** for the frontend. The platform integrates with an **SQLite** database to manage users, restaurants, menus, orders, and cart items. Below is a breakdown of the architecture and functionality:

#### **Backend Implementation**

1. **Flask Framework**:
   * Flask was used to handle HTTP requests and responses. The platform consists of routes that correspond to various pages such as home, login, cart, checkout, and restaurant menus.
   * Flask’s session management system is used to maintain the state of logged-in users. Users are required to log in to place an order or view their cart.
2. **Database**:
   * The app interacts with an SQLite database to interact with the database, allowing for operations such as fetching restaurant details, adding items to the cart, and placing orders.
   * Tables include Restaurant, MenuItem, Order, and OrderItem, among others.
3. **Key Features**:
   * **User Authentication**: Users log in using their email and password, and their session is maintained across requests. If they are not logged in, they are redirected to the login page.
   * **Order Management**: Users can add items to their cart, remove them, and proceed to checkout. Once an order is confirmed, it is stored in the database, and the order status is updated.
4. **Routes and Views**:
   * /home: Displays a list of restaurants fetched from the database. Users can browse restaurant details and view their menus.
   * /restaurant/<restaurant\_id>: Displays the menu of a selected restaurant. Users can add items to their cart.
   * /cart: Displays the items currently in the user's cart. Users can remove items or proceed to checkout.
   * /checkout: Displays a summary of the cart with the option to confirm the order.
   * /order-success: Displays a success message after an order is placed.

#### **Frontend Implementation**

The frontend is built using **HTML** and **CSS**, and it renders dynamic content using **Flask's templating engine**. Below are the key templates:

1. home.html:
   * Displays a list of restaurants available for delivery. Each restaurant is shown with its name, description, rating, and an image.
   * A search bar allows users to search for restaurants or dishes.
2. restaurant\_menu.html:
   * Displays the menu for a specific restaurant, showing the items along with their names, descriptions, and prices. Users can add items to their cart from this page.
3. cart.html:
   * Shows the items currently in the user's cart with details such as quantity and total price. Users can remove items from their cart or proceed to checkout.
4. checkout.html:
   * Displays a summary of the cart, showing the items and their total price. Users can confirm the order by submitting the form.
5. checkout\_success.html and order\_success.html:
   * These pages display a confirmation message once an order is successfully placed, thanking the user for their purchase and offering a link back to the home page.
6. login.html:
   * Displays the login form for users to sign in with their email and password. Upon successful login, users are redirected to the home page or the page they were trying to access.
7. account.html:
   * Shows the user's account details (name, email, phone number) and provides an option to log out.

#### **Flow of Operation**

1. User Login:
   * Users must log in to access their account and place orders. If they are not logged in, they are redirected to the login page.
2. Restaurant and Menu Browsing:
   * After logging in, users can browse available restaurants on the home page, view their menus, and select items to add to their cart.
3. Cart Management:
   * Users can view, add, or remove items from their cart. The cart displays the items along with the total price.
4. Checkout and Order Placement:
   * After reviewing the cart, users can confirm their order by proceeding to checkout. Once confirmed, the order is placed and stored in the database, and the user receives a success message.
5. Order Confirmation:
   * After successfully placing an order, the user is shown a confirmation page with a thank-you message and a link to return to the homepage.

### 

### **Physical Database Design**

### **1. Indexing**

Indexing is used to improve query performance by creating data structures that allow fast data retrieval without scanning the entire table. For the food delivery platform:

* IndividualUser Table:
  + An index is created on Email to speed up user authentication queries.
  + Another index is added on UserType to allow efficient filtering for customers and delivery personnel.
* Address Table:
  + An index on State and City enables faster searches for delivery addresses based on geographic location.
* Restaurant Table:
  + Indexes on Rating and CreatedAt facilitate quick sorting and filtering for highly rated restaurants or recently added restaurants.
* Payment Table:
  + Indexes on PaymentMethod and PaymentStatus improve the performance of payment-related queries.

### **2. Partitioning**

Partitioning divides a large table into smaller, more manageable chunks, improving query performance and maintenance.

* Address Table:
  + Partitioned by State using LIST partitioning. For example, states like NY, CA, and TX can have their own partitions. This minimizes the scope of searches when querying data for specific states or regions.
* Order-Related Data:
  + Tables like Order and OrderItem could benefit from RANGE partitioning based on OrderDate to segregate historical and recent orders.

### **3. Clustering**

Clustering reorganizes a table's physical storage to group related data together. This reduces the number of I/O operations for frequently queried data.

* IndividualUser Table:
  + Clustered by UserType, separating customers and delivery personnel into distinct groups within the table. This is beneficial for queries targeting a specific user type (e.g., fetching delivery personnel availability).
* MenuItem Table:
  + Can be clustered by MenuID to group items belonging to the same menu for optimized retrieval during menu queries.

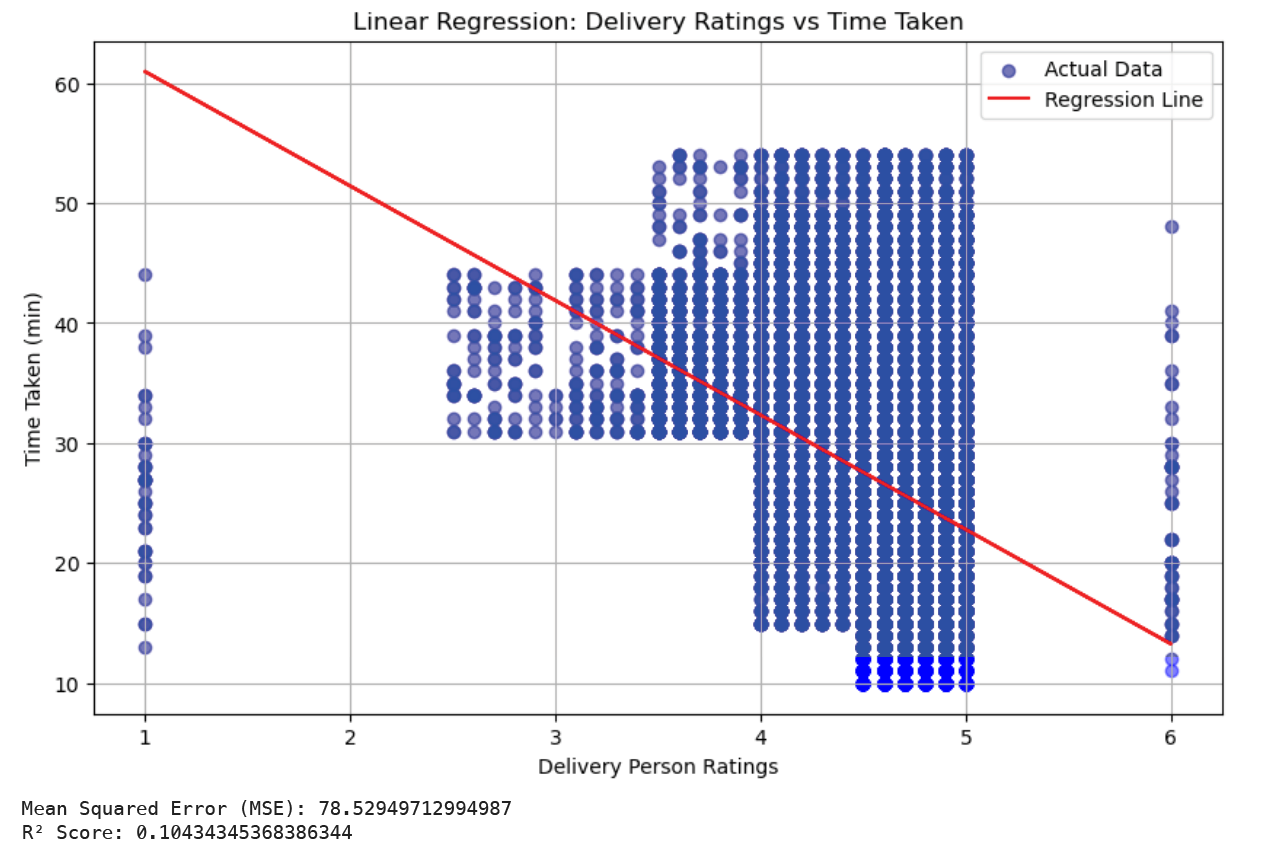
## 

## **Machine Learning to Provide Business Insights**

Machine learning (ML) provides actionable business insights for our food delivery platform, helping to optimize operations, enhance customer satisfaction, and drive efficiency.

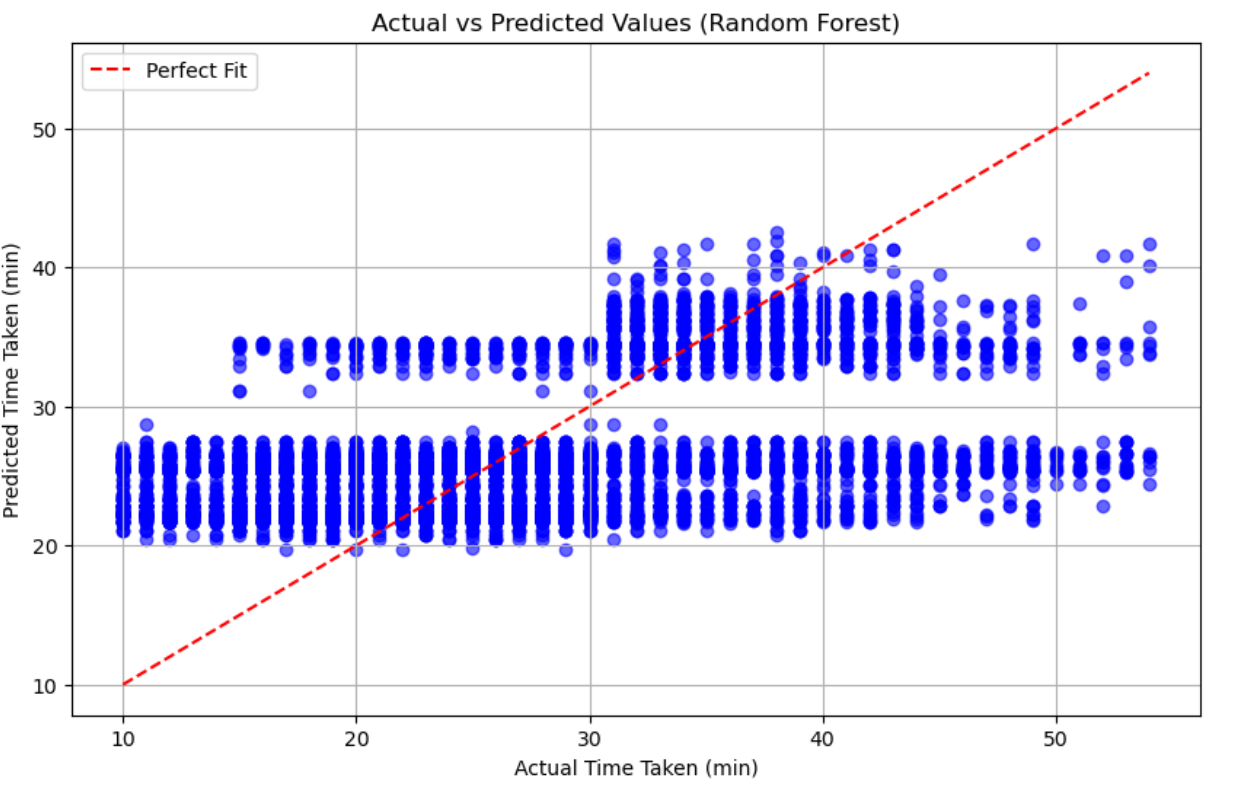
### **1. Customer Satisfaction Analysis**

* Faster deliveries correlate with higher ratings, emphasizing the need for logistical efficiency.
* **Business Implications**:
  + Invest in route optimization and faster vehicles.
  + Highlight estimated delivery times during order placement.



**2. Delivery Time Prediction**

* **Factors**: Delivery person ratings, order type, and vehicle type and delivery times significantly influence the delivery times.
* **Business Implications**:
  + Assign time-sensitive orders to highly rated personnel.
  + Use faster vehicles like motorcycles for urban deliveries.
  + Set realistic customer expectations based on predictions.



### **3. Operational Recommendations**

* Train delivery personnel on route optimization to improve efficiency and ratings.
* Forecast demand patterns to allocate resources effectively.

### **4. Feature Enhancements**

* **Real-Time Predictions**: Incorporate live traffic data for dynamic delivery estimates.
* **Advanced Models**: Use deep learning for improved demand forecasting and efficiency.
* **Expanded Features**: Factor in weather and regional data for better accuracy.

### 

### 

# 

# **Data Governance**

### **1. Data Quality Management**

* Ensure Accuracy and Consistency: Standardize data formats and validate at entry.
* Implementation: Regular audits and automated validation processes.

### **2. Prevention of Data Loss and Leakage**

* Encryption and Access Control:
  + Encrypt data in transit and at rest.
  + Use role-based access control (RBAC) and regular permission reviews.
* Backup and Recovery:
  + Regular backups with tested recovery procedures.

### **3. Data Lifecycle Management**

* Collection and Storage:
  + Collect only necessary data and store using scalable cloud solutions.
* Retention and Deletion:
  + Automate data retention policies and deletion processes.

### **4. Safeguarding Against Bias**

* Evaluation and Mitigation:
  + Regularly check for biases in ML models.
  + Retrain models with diverse datasets and fairness constraints.

### **5. Fairness, Accountability, and Transparency**

* Fair Practices:
  + Prevent discriminatory practices in system outputs.
* Transparency:
  + Communicate data use clearly and document ML model decisions.

### 

### 

### 

## 

## **Future Improvements**

### **1. Scalability:**

* Deploy application to Azure for higher availability.
* Introduce load balancing for API endpoints.

### **2. Advanced Analytics:**

* Use real-time analytics for demand forecasting.
* Enhance ML models with deep learning techniques for better predictions.

### **3. Feature Expansion:**

* Add support for multiple delivery zones.
* Enable personalized recommendations systems based on user preferences.